ABC INFORMATIQUE

N° 89

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE

Analyse du Spectrum

Les clubs Microtel

Bananarama

Commandes transitoires

EDITIONS ATLAS

M 6062-89-12,50 F 90 FB - 3,80 FS - $ 1.95

Page manquante
(publicité et colophon)

Scout et B-Star

Tout en examinant de près la planification stratégique en intelligence artificielle, nous allons étudier certaines améliorations ainsi que d'autres stratégies requises par les jeux de hasard.

La procédure alpha-bêta, que nous avons déjà vue, constitue une grande amélioration pour l'opération de « minimaximalisation » (puisqu'elle identifie et supprime les branchements redondants de l'arbre de jeu); elle est au cœur des programmes de jeux d'échecs sur ordinateur depuis de nombreuses années. Mais, récemment, deux autres stratégies ont été proposées. L'une est l'algorithme Scout de Judea Pearl et l'autre est l'algorithme B-Star (B*) de Hans Berliner.

L'essentiel de la méthode Scout consiste à posséder une fonction d'évaluation très fine pouvant servir à rejeter les mouvements non plausibles sans effectuer plus de recherches. Seuls les mouvements qui semblent possibles doivent être examinés soigneusement.

La méthode B* analyse à fond les mouvements au niveau supérieur de l'arbre et essaie, aussi rapidement que possible, d'effectuer l'une des choses suivantes :

- Prouver que le mouvement *apparemment* le meilleur est bien le meilleur possible.
- Prouver qu'aucun autre mouvement n'est meilleur.

Cette double stratégie est mise en œuvre par une paire de procédures nommées ProveBest et RefuteBest, pour lesquelles deux valeurs doivent être attribuées à chaque nœud de l'arbre — une évaluation optimiste, une autre pessimiste. L'objectif est d'amener l'algorithme à se concentrer dans des zones de l'arbre du jeu où il y a incertitude et où cette incertitude peut modifier la décision finale.

Il serait faux de croire que la recherche arborescente est la seule approche dans les jeux sur ordinateur. Il existe certains jeux intéressants où la philosophie de recherche semble ne pas aboutir, quelles que soient les astuces adoptées pour accélérer le processus. Ainsi, nous pouvons songer à de nombreux jeux de cartes très populaires (comme le bridge et le poker) et plusieurs jeux sur damier (comme le go et le go-moku).

Ces jeux peuvent accepter divers niveaux de maîtrise; ils font réellement appel à l'intelligence. Jusqu'ici, toutes les tentatives visant à les pro-

Recherche futile
Il existe de nombreux jeux pour lesquels les stratégies de recherche ne sont pas efficaces; soit le jeu comporte un élément aléatoire, comme le backgammon, soit l'arbre du jeu effectue rapidement des branchements sur des mouvements successifs pour produire un nombre immense de permutations de mouvements possibles. Pour créer des programmes qui peuvent exécuter de tels jeux, d'autres méthodes doivent être trouvées pour remplacer la recherche arborescente. (David Lawrence.)

Jargon de recherche arborescente

Arbre de jeu	Structure arborescente formée en tenant compte des mouvements possibles et des réponses possibles de l'adversaire, et ainsi de suite.
Couche	Un niveau dans l'arbre de jeu.
Prévision	Processus de constitution de l'arbre de jeu.
Valeur de référence	Valeur attribuée à un nœud dans l'arbre de jeu par l'examen des valeurs inférieures, en partant du bas.
Minimaximalisation	Choisir quelle valeur fournir à l'arbre comme référence en minimisant les couches impaires (autre) et en maximisant les couches paires (soi).
Algorithme Alpha-Bêta	Un perfectionnement de la minimaximalisation qui, éliminant des portions de l'arbre de jeu, pourraient modifier le résultat au niveau supérieur.
Facteur de branchement	Le nombre moyen de branchements ou de mouvements à chaque niveau de l'arbre de jeu. Le jeu oriental go a un facteur de branchement de plus de 200.

grammer avec une recherche arborescente n'ont pas abouti. L'une des raisons est que le facteur de branchement est simplement trop grand et génère un trop grand nombre de combinaisons de mouvements possibles que l'ordinateur ne peut pas gérer simultanément.

Une autre réponse est que les algorithmes de recherche arborescente ne fournissent qu'une approximation grossière de la façon qu'ont les humains à attaquer des problèmes : seuls, les experts recherchent mentalement dans leurs propres arbres de jeux (construits de façon interne dans des circonstances spéciales) et encore d'une manière pas très efficace.

Hans Berliner, qui a conçu la méthode B*, a mis au point un jeu de backgammon qui a battu le champion du monde au cours d'un match en 1980. Mais le programme n'effectue aucune recherche, et en tout cas pas dans le sens conventionnel. Si vous vous rappelez comment se joue le backgammon, vous remarquerez que son arbre de jeu introduit des probabilités : le dé crée des branchements qui sont contrôlés par chaque joueur, ce qui rend très difficile la mise au point de procédures de prévision. Le programme de Berliner est très efficace pour calculer les jets probables de dés et les diverses combinaisons de pièces qui se produisent au cours du jeu.

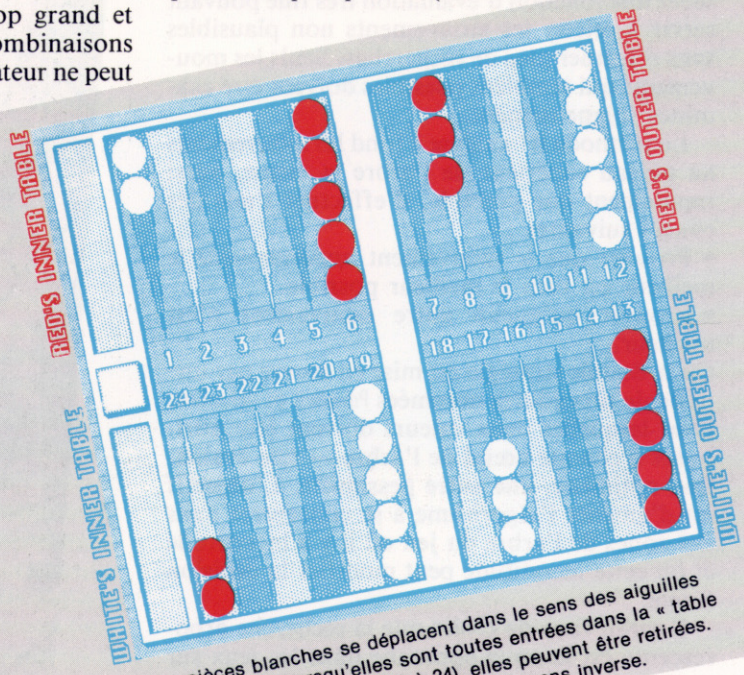
Le go est un jeu oriental dans lequel des pierres se déplacent sur une grille de 19 lignes horizontales et de 19 lignes verticales. L'objectif est d'entourer des zones de la grille pour gagner de l'espace (territoire) tout en retirant de la partie le maximum de pierres de son adversaire. L'élément de chance est absent, mais le facteur de probabilité est si grand que les techniques reposant sur la recherche sont inutiles.

Les meilleurs programmes de go traduisent les éléments du damier en termes d'unités plus élevées que de simples pierres (comme des « chaînes » ou des « armées ») qui constituent des groupements significatifs pour l'œil humain; la programmation du go a probablement pris du retard par rapport aux échecs parce que notre perception humaine est inadéquate. Les Japonais, qui vénèrent ce jeu, considéreront-ils le go comme un projet approprié pour leurs machines de la cinquième génération? Un tel programme, réussi, atteindra certainement les limites des notions d'intelligence artificielle.

Des programmes de go existent déjà pour la génération d'ordinateurs domestiques actuelle; nous examinerons ultérieurement les problèmes impliqués dans la programmation d'un tel jeu. Cepen-

La table interne

Le backgammon se joue sur une tablette formée de vingt-quatre triangles sur lesquels les pièces sont déplacées, selon le jet d'une paire de dés. L'objectif du jeu est que toutes vos pièces fassent le tour de la tablette avant celles de votre adversaire.
Si les nombres qui apparaissent sur les dés sont différents, vous pouvez soit déplacer une pièce du nombre total de points indiqués sur les dés, soit déplacer deux pièces séparément selon chaque nombre obtenu. Lorsqu'une pièce est déplacée selon la somme des dés, elle effectue en fait deux mouvements distincts et « touche » le point intermédiaire entre les deux mouvements.
Les points occupés par deux pièces ou plus (hommes) d'une couleur ne peuvent être déplacés ou touchés par le côté adverse. Si un seul homme occupe un point, et s'il est touché par l'adversaire, il sort de la tablette et doit être entré de nouveau par son propriétaire avant de pouvoir effectuer un autre mouvement. L'illustration présente un mouvement dans le jeu entre le programme Backgammon de Hans Berliner et le champion du monde, Luigi Villa, qui eut lieu à Monte-Carlo en 1980.

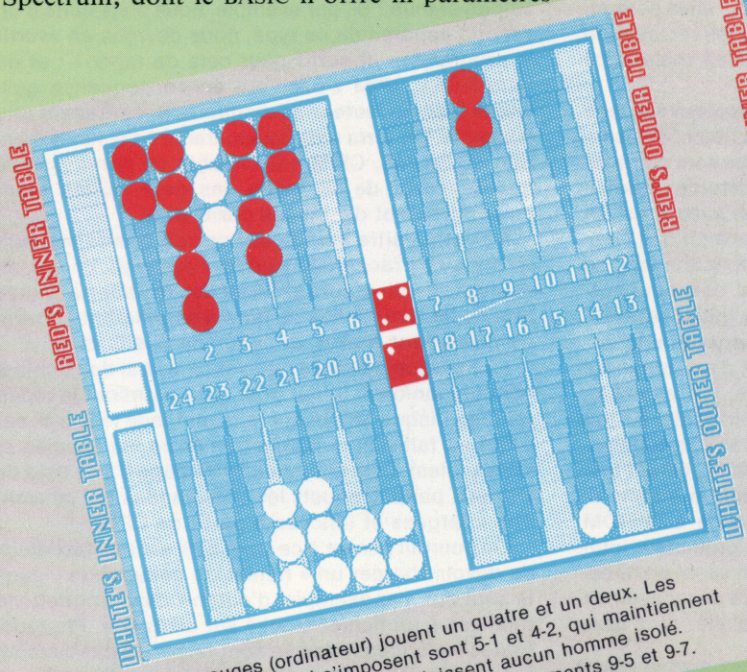


Les pièces blanches se déplacent dans le sens des aiguilles d'une montre. Lorsqu'elles sont toutes entrées dans la « table interne » blanche (points 19 à 24), elles peuvent être retirées. Les pièces rouges se déplacent dans le sens inverse.

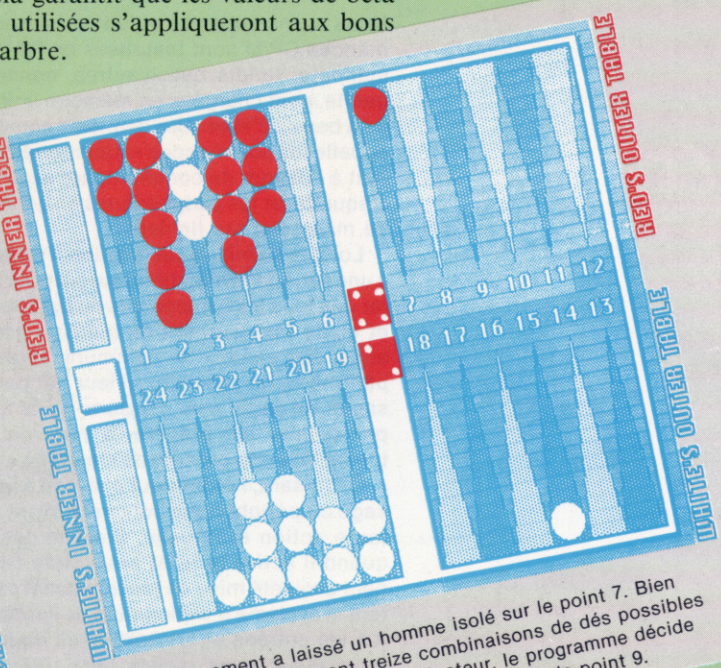
dant, on est encore loin de pouvoir espérer que ces programmes vaincront les champions du jeu.

Pour illustrer les notions importantes de la recherche arborescente, nous avons conçu un jeu artificiel qui est une recherche pure. Celui-ci a été donné dans un article précédent, et utilisait massivement des fonctions récursives avec des paramètres. Dans ce jeu, vous et l'ordinateur sélectionnez à tour de rôle l'une des quatre fonctions qui modifient une valeur existante pour en produire une nouvelle. Vous tentez de réduire la valeur à -255 tandis que l'ordinateur essaie de l'augmenter à 255. A chaque étape, l'ordinateur utilise le minimaximum alpha-bêta pour choisir quelle fonction est la plus indiquée.

Pour les utilisateurs du Commodore 64 et du Spectrum, dont le BASIC n'offre ni paramètres ni variables locales, nous répétons le programme dans une version qui n'utilise que des GOSUB. Les paramètres et les variables locales (sauf D, le compteur de profondeur) ont été remplacés par des tableaux, déclarés à la ligne 1100. D remplit maintenant le rôle de pointeur de pile, suivant l'évolution des éléments actuellement accessibles. Ces deux routines diffèrent principalement en ce que tout doit être indexé par D, particulièrement A(), servant à contenir la meilleure valeur trouvée jusqu'ici, et B(), servant à contenir la plus mauvaise. Cela garantit que les valeurs de bêta et de gamma utilisées s'appliqueront aux bons niveaux de l'arbre.



Les rouges (ordinateur) jouent un quatre et un deux. Les mouvements qui s'imposent sont 5-1 et 4-2, qui maintiennent chaque point couvert et ne laissent aucun homme isolé. L'ordinateur choisit à la place les mouvements 9-5 et 9-7.



Le mouvement a laissé un homme isolé sur le point 7. Bien que Villa ait maintenant treize combinaisons de dés possibles pouvant retirer la pièce de l'ordinateur, le programme décide de ne pas risquer de laisser deux pièces sur le point 9.

Le jeu des nombres

```
80 GOSUB 1000:REM INITIALISATION
90 GOSUB 1600:REM INSTRUCTIONS
100 :
110 REM **** BOUCLE PRINCIPALE DU PROGRAMME ****
120 GOSUB 2000:REM PREPARER NOUVEAU JEU
130 INPUT "QUI COMMENCE (1=VOUS, 2=MOI)" ;H1
140 IF H1<1 OR H1 2)THEN 130
150 REM **** BOUCLE DU JEU ****
160 IF H1=1 THEN GOSUB 3000
170 REM ** TOUR DU JOUEUR **
180 GOSUB 3500:REM AFFICHAGE DU DAMIER
190 H1=1:REM TOUJOURS 1 APRES PREMIER CYCLE
200 GOSUB 4000:REM TESTE POUR VICTOIRE
210 IF EG=0 THEN GOSUB 5000
220 REM ** TOUR DE L'ORDINATEUR **
230 GOSUB 3500:REM AFFICHER ETATS DU JEU
240 GOSUB 4000:REM TEST FIN DE JEU
250 IF EG=0 AND M<=33 THEN 150:REM BOUCLE RETOUR
260 REM **** FINALE ****
270 GOSUB 6000:REM FELICITATIONS
280 INPUT "AUTRE JEU (1=OUI, 2=NON)" ;Y
290 IF Y<1 OR Y>2 THEN 280
300 IF Y=1 THEN 110:REM NOUVEAU JEU
310 PRINT:PRINT "AUREVOIR ET MERCI"
320 END
330 :
500 REM **** MAXIMISER ****
510 D=D+1:C1=C1+1
520 IF D)=MD OR ABS(V(D))>HI THEN A(D)=V(D):D=D-1:RETURN
530 REM ** PLUS PROFONDEMENT **
540 P(D)=0
550 REM ** DANS ARBRE **
560 P(D)=P(D)+1:H=P(D):V=V(D):GOSUB 5500:REM DEPLACER
570 IF D=1 THEN PRINTCHR$(64+H);" = ";
580 D1=D+1
590 A(D1)=A(D):B(D1)=B(D):V(D1)=V:GOSUB 700:REM MINIMALISER
    APPEL
600 IF B(D+1) A(D))THEN A(D)=B(D+1):K(D)=P(D)
610 IF D=1 THEN PRINTB(D+1);"; "
620 IF P(D)<=3 AND A(D)<B(D) THEN 550
630 IF D=1 THEN BV=A(D):HH=K(D):REM GARDER MEILLEUR JUSQU'ICI
640 D=D-1:RETURN
650 :
700 REM **** MINIMISER ****
710 D=D+1:C2=C2+1
720 IF D)=MD OR ABS(V(D))>HI THEN B(D)=V(D):D=D-1:RETURN
730 P(D)=0
740 REM ** DANS ARBRE **
750 P(D)=P(D)+1:H=P(D):V=V(D):GOSUB 5500:REM DEPLACER
760 D1=D+1:A(D1)=A(D):B(D1)=B(D):V(D1)=V
770 GOSUB 500:REM MAXIMALISER APPEL
780 IF A(D+1)<B(D) THEN B(D)=A(D+1)
790 IF P(D)<=3 AND B(D) A(D) THEN 740
800 D=D-1:RETURN
810 :
1000 REM **** INITIALISER ****
1010 BL$=""
1020 REM ** DEFINITIONS DES QUATRE FONCTIONS **
1030 DEF FNA(X)=2*X-7
1040 DEF FNB(X)=INT(X/2)+1
1050 DEF FNC(X)=4*X+17
1060 DEF FND(X)=3*X-4
1070 LO=-255:HI=255
1080 REM ** TABLEAUX UTILISES PAR MINIMAX **
1090 D=16
1100 DIM V(D),A(D),B(D),P(D),K(D)
1110 RETURN
1120 :
1600 REM **** INSTRUCTIONS **
1610 PRINT "BIENVENUE AU JEU DES NOMBRES"
1620 PRINT "J'ESSAIE DE MAXIMALISER"
1630 PRINT "VOUS DEVEZ MINIMALISER"
1640 PRINT "pour un exemple de MOUVEMENT tapez!
1650 PRINT "A, B, C OU D.TAPEZ X POUR LE FAIRE"
1660 PRINT:RETURN
1670 :
2000 REM **** PREPARATION ****
2010 M=0:V=INT(RND(1)*15)-8:REM ETAT INITIAL
2020 EG=0
2030 PRINT" ETAT INITIAL = ";V
2040 RETURN
2050 :
3000 REM **** MOUVEMENT DU JOUEUR ****
3010 M=M+1:PRINT
3020 PRINT VOTRE DEPLACEMENT EST ;
3030 INPUT H$
3040 IF H$="A" THEN PRINT FNA(V):H=1
3050 IF H$="B" THEN PRINT FNB(V):H=2
3060 IF H$="C" THEN PRINT FNC(V):H=3
3070 IF H$="D" THEN PRINT FND(V):H=4
3080 IF H$<>"X" THEN 3020:REM MOUVEMENT NON CHOISI
3090 GOSUB 5500:REM FAIRE MOUVEMENT
3100 RETURN
3110 :
3500 REM **** AFFICHER DAMIER ****
3510 PRINT:PRINT"     MOVE";M;" -->";
3520 IF M<1 THEN RETURN
3530 PRINT CHR$(64+H);
3540 PRINT" = ";V:PRINT:RETURN
3550 :
4000 REM **** GAGNER TEST ****
4010 IF M<1 THEN RETURN
4020 EG=0
4030 IF V<LO THEN EG=-1
4040 IF V>HI THEN EG=1
4050 RETURN
4060 :
5000 REM **** MOUVEMENT DE L'ORDINATEUR ****
5010 W=V: REM SAUVEGARDER ETAT EN COURS
5020 M=M+1
5030 MD=6:REM PROFONDEUR MAY
5040 IF M<4 THEN MD=4
5050 IF M>8 THEN MD=8
5060 GOSUB 5200:REM --->H
5070 V=W:REM REMETTRE ETAT
5080 GOSUB 5500 :REM DEPLACER
5090 RETURN
5100 :
5200 REM **** SELECTION DE MOUVEMENT ****
5210 BV=LO:D=0
5220 V(1)=V:A(1)=LO:B(1)=HI
5230 GOSUB 500:REM MAXIMALISER
5240 H=HH
5250 PRINT:INPUT "APPUYEZ SUR RETURN POUR CONTINUER" ;Q
5260 RETURN
5270 :
5500 REM **** FAIRE DEPLACEMENT ****
5510 IF H=1 THEN V=FNA(V):RETURN
5520 IF H=2 THEN V=FNB(V):RETURN
5530 IF H=3 THEN V=FNC(V):RETURN
5540 IF H=4 THEN V=FND(V):RETURN
5550 :
6000 REM **** FELICITATIONS ****
6010 PRINT:PRINT FIN DE PARTIE
6020 IF EG)0 THEN PRINT "J'AI GAGNE"
6030 IF EG<0 THEN PRINT "VOUS AVEZ GAGNE"
6040 IF EG=0 THEN PRINT "PARTIE NULLE"
6050 RETURN
```

Commandes en transit

Poursuivant notre étude du système CP/M, nous allons maintenant examiner les commandes « transitoires », qui se révèlent fort utiles lors de la gestion des fichiers et des périphériques.

Nous avons constaté qu'un certain nombre de commandes CP/M sont installées en RAM de façon permanente, tandis que d'autres, conservées sur disquette, sont chargées en mémoire uniquement quand on a besoin d'elles. Ces dernières constituent ce qu'on appelle les commandes « transitoires » : elles permettent à l'usager de communiquer avec les lecteurs de disquette (et les périphériques en général), mais aussi de manipuler les fichiers.

Lors de sa mise sous tension, l'ordinateur procède à une série de vérifications qui ont pour objet de s'assurer que tous les composants du système sont présents et qu'ils fonctionnent de manière correcte. Il ouvre aussi des canaux de communication avec tous les périphériques présents : il leur fait parvenir des messages, en partie pour s'assurer qu'ils fonctionnent, en partie pour les préparer à recevoir des données. L'ensemble du processus s'appelle « initialisation ».

Ce faisant, l'ordinateur s'attend à recevoir un message du périphérique concerné avant d'entreprendre toute action éventuelle. C'est le cas, par exemple, quand il active la ROM BASIC : une cartouche de jeu peut avoir été mise en place, auquel cas elle supplantera le système d'exploitation de l'appareil, gérant toutes les entrées et les sorties en direction du microprocesseur, sans jamais faire référence à la ROM BASIC. Dans le même ordre d'idées, un ordinateur tournant sous CP/M, après avoir ouvert un canal en direction d'un lecteur de disquette, attendra de recevoir un message de ce périphérique avant d'aller plus loin.

Extension	Commentaire	Exemple
ASM	Pour fichier source en assembleur	CODEPROG.ASM
BAK	Copie fichier texte créé par éditeur	MEMO.BAK
BAS	Suit un fichier source en BASIC	PROG.BAS
COM	Extension de commande ou fichier transitoire	PIP.COM
HEX	Fichier hexadécimal en langage machine	GRAPH.HEX
INT	Pour un programme BASIC compilé	JEU.INT
PRN	Pour listages programmes assembleur	CODEPROG.PRN
SUB	Fichier pour exécution commandes par lot	REPART.SUB
$$$	Fichier temporaire créé par l'éditeur	TEL.$$$

L'initialisation d'un lecteur de disquette a pour effet de contraindre sa tête de lecture/écriture à lire la première piste du disque sous système CP/M. Elle tournera en rond aussi longtemps qu'une disquette ne sera pas en place. La piste zéro contient un programme « chargeur », qui, à son tour, enjoint à l'ordinateur de commencer à charger en mémoire le reste de CP/M. L'absence d'un programme de ce type provoque l'apparition d'un message d'erreur.

Dès que CP/M est installé en mémoire, on voit apparaître à l'écran un curseur clignotant placé juste après l'expression A> (0A> sur certaines machines). Cela signifie que le système est prêt à accepter vos commandes, qu'il fait usage du lecteur de disquette A et qu'il attend que vous écriviez ou lisiez des informations. A est le périphérique dont CP/M fait usage « par défaut » (à partir de l'instant où il est lancé). Pour le moment, c'est bien le cas.

Que se passera-t-il, cependant, si nous avons plus d'un lecteur de disquette, et que nous voulons les employer tous ? Si, par exemple, nous disposons d'un second appareil de ce type, nous devrons en avertir le programme. Il suffit pour cela de taper B : et de faire RETURN. CP/M s'assurera en conséquence qu'il existe bien un lecteur de disquette B, contenant une disquette. On verra ensuite apparaître à l'écran la mention B>. En fait, CP/M est capable de gérer jusqu'à quatre unités de ce genre, les deux autres étant respectivement désignées par C et D.

Il peut paraître surprenant de voir quatre lecteurs de disquette raccordés à un seul appareil. En fait, de nombreux ordinateurs utilisent des modèles à double tête de lecture. Celles du haut s'appelleront conventionnellement A et B, et celles du bas C et D.

L'opération suivante consiste à voir si des fichiers sont disponibles. Il faut pour cela examiner le répertoire du disque en faisant la commande DIR (ou dir, car CP/M ne fait pas la distinction entre majuscules et minuscules). Cela provoque l'affichage d'une liste de fichiers, parmi lesquels les commandes qui peuvent être chargées et lancées à partir de CP/M.

On pourrait penser à ce sujet qu'il est très fastidieux de devoir charger un « fichier de commande » dans le seul but, par exemple, d'obtenir des informations relatives à un fichier stocké sur disquette. Pourquoi toutes les commandes ne sont-elles pas directement placées en RAM, ce qui permettrait de les mettre en œuvre à volonté ? Nous avons déjà indiqué que cela économisait de l'espace mémoire, mais il y a une autre raison : favoriser la compatibilité entre systèmes différents.

CP/M est en effet pourvu de plusieurs commandes qui rendent possible la gestion de fichiers contenus sur disquettes. Mais les constructeurs n'ont jamais pu s'entendre sur un système standard : il varie d'un ordinateur à l'autre. Pourtant, le but même de CP/M est d'être portable. Il s'ensuit qu'il est par exemple dépourvu d'une commande de formatage spécifique ; elle fera partie des commandes transitoires propres à tel ou tel appareil, alors que chacun fonctionne à sa manière.

Nous avons déjà vu qu'un fichier CP/M se compose de deux parties. Son nom se décompose en un nom « primaire » suivi d'un point et d'une extension. Celle-ci détermine la façon dont il sera chargé en mémoire. Le « disque système » accueille un certain nombre de fichiers affectés aux commandes transitoires, toutes suivies de l'extension .COM. Une fois installées en RAM, elles seront lancées automatiquement.

La commande stat

Pour charger un fichier de commande, il suffit d'en taper le nom primaire et de faire RETURN. L'exécution est automatique. Des fichiers pourvus d'extensions

différentes ne se comportent pas ainsi : nous les passerons en revue ultérieurement.

Nous vous avons présenté la commande DIR. Il existe une autre commande transitoire, nommée STAT, qui lui est apparentée : elle vous donne des informations supplémentaires sur le disque et les fichiers qu'il renferme. Elle précise l'espace mémoire restant disponible, aussi bien dans l'ordinateur que sur la disquette, mais aussi les opérations de lecture et d'écriture qui sont autorisées. C'est ainsi que R/W signale que vous pouvez faire les deux (READ/WRITE : lire et écrire), tandis que R/O vous avertit que vous ne pourrez que lire le disque, qui est donc protégé.

STAT permet de ce point de vue de « verrouiller » la disquette : il faut pour cela taper STAT D : R/O (D correspondant à DRIVE, « lecteur », doit être remplacé par le caractère affecté à tel ou tel d'entre eux). Désormais, toute tentative d'écriture se soldera par un message d'erreur. STAT affiche par ailleurs toutes les extensions de fichiers, ainsi que le nombre de secteurs affectés au fichier logique par enregistrement. Il est même possible de connaître, grâce à STAT, la taille de fichiers individuels — tapez STAT suivi du nom de celui qui vous intéresse. Enfin STAT peut être mis en œuvre pour examiner, et au besoin changer, le statut des périphériques raccordés à l'ordinateur. STAT DEV (pour DEVICE, « appareil ») donne par exemple leur liste complète, écran compris.

Un système d'exploitation vraiment utile doit permettre le transfert d'un fichier d'une disquette à l'autre. C'est une opération dont la commande PIP se charge. C'est une abréviation de *Peripheral Interface Program* (gestionnaire de périphérique). Comme ce nom l'indique, PIP est une commande très souple d'emploi qui ne se limite pas à la copie mais favorise l'envoi des fichiers vers une imprimante ou tout autre périphérique.

Il vous faudra d'abord charger le fichier de commande (tapez PIP et faites RETURN). Notez l'apparition à l'écran d'un astérisque, qui indique que PIP est déjà lancé et attend la commande suivante. Le format de base est le suivant : D:NOMCOPIE = D : NOMORIGINE. Supposons que vous ayez sur une disquette un fichier texte intitulé LIVRE.TXT, et que vous désiriez le recopier sur une autre, en lui donnant le nom ROMAN.TXT. La manœuvre sera donc la suivante : enlever le disque système du lecteur, placer la disquette contenant LIVRE.TXT dans le lecteur A, et l'autre dans le lecteur B, et taper B : ROMAN.TXT = A : LIVRE.TXT.

Il est à noter que le lecteur « objet » (celui qui effectuera la copie) doit être placé *avant* le lecteur « source » (qui contient le fichier original). Par ailleurs, lorsqu'on n'a pas affaire à une commande, le nom du fichier doit être entré en entier, extension comprise. Une fois l'opération menée à bien, il est possible de vérifier qu'elle s'est bien passée en étudiant le répertoire du lecteur B.

Il est toujours possible de faire usage de PIP avec un seul lecteur, mais, bien entendu, celui-ci tiendra lieu à la fois de « source » et d' « objet », ce qui signifie que vous devrez procéder en cours d'opération à des échanges périodiques de disquettes. CP/M vous préviendra à chaque fois, après avoir recopié des secteurs du fichier original, de façon qu'ils soient transférés sur la copie.

PIP sert aussi à envoyer notre fichier LIVRE.TXT en direction de l'imprimante. Le format de base est dans ce cas PIP LPT:=B:LIVRE.TXT. LPT correspond à « Line PrinTer » (ligne imprimante). Il est à noter que, dans ce cas particulier, PIP n'est pas chargé séparément du reste de la commande. C'est tout à fait possible sous CP/M, PIP n'étant pas une commande qu'il faille placer en mémoire avant que quoi que ce soit d'autre puisse être entrepris. LPT est rendu indispensable du fait que le périphérique de destination n'est pas un lecteur de disquette mais une imprimante. Enfin, nous avons spécifié le caractère du lecteur contenant LIVRE.TXT. Si ce lecteur est, au moment de la mise en œuvre, le lecteur B, cette précision sera même superflue, et certaines versions de CP/M n'en tiennent aucun compte, même quand c'est le lecteur A qui est activé. En effet CP/M examine successivement tous ceux qui lui sont raccordés avant d'afficher le message FILE NOT FOUND (fichier absent).

Nous avons aussi, en transférant le fichier d'une disquette à l'autre, changé son nom. Cela n'est nullement obligatoire, et la copie peut garder le même nom que l'original. Supposons toutefois que nous désirions modifier le nom d'un fichier, sans pour autant avoir à le copier. Nous ferions alors usage de la commande REN (pour RENAME : donner un nouveau nom). Si donc nous voulons que ROMAN.TXT devienne LIVRE.TXT, nous taperons REN LIVRE.TXT = ROMAN.TXT, en prenant bien garde de placer en premier le titre « objet ». Le répertoire contiendra alors un nouveau nom de fichier, tandis que l'ancien sera supprimé.

A ce sujet, un fichier peut être effacé sur une disquette à l'aide de la commande ERA (pour ERASE, effacer). Ses effets sont redoutables, et il faut donc prendre des précautions, afin de prévenir une erreur toujours possible, et qui se révélerait catastrophique. CP/M doit savoir à quelle disquette vous faites allusion, faute de quoi il pourrait s'en prendre à une version du fichier qui n'est pas la bonne ! C'est pourquoi il est toujours judicieux, même si ce n'est pas absolument indispensable, de préciser avec quel lecteur on travaille. On aura ainsi, par exemple, ERA B : LIVRE.TXT.

Nous nous intéresserons ultérieurement à d'autres commandes du programme, à l'emploi des extensions, et verrons de près certains des caractères de contrôle utilisés par CP/M.

ASM : l'Assembleur du CP/M

Dans la plupart de ses versions, CP/M comporte son propre assembleur, appelé ASM. Bien que fréquemment négligé par les programmeurs, c'est un programme très puissant, qui, en plus des commandes traditionnelles, permet aussi l'assemblage conditionnel. Son seul gros défaut est d'avoir été prévu à l'origine pour le microprocesseur 8080, qui a plus tard cédé la place au Z80. Le code objet destiné au premier tournera bien sûr avec le second, puisqu'ils sont compatibles, mais certains mnémoniques du Z80 ne sont pas acceptés par ASM. Vous en trouverez la liste détaillée dans l'ouvrage de Rodnay Zaks publié aux éditions Sybex, *La programmation du Z80*. Si, en revanche, vous comptez faire un usage régulier de cet assembleur sur un ordinateur équipé d'un Z80, les choses seront plus compliquées.

Les clubs Microtel

Elle ne donne pas dans le spectacle et, pourtant, elle ne compte pas moins de deux cent soixante clubs en France. Il s'agit de l'association Microtel qui met l'informatique à la portée de tous.

Avec l'arrivée de l'ordinateur à l'école, et des centres X2000 largement équipés, le rôle de la fédération des clubs Microtel, très bien implantés partout en France, semble devoir se déplacer vers une plus grande ouverture et un plus grand professionnalisme. Les clubs Microtel accueillent pour une cotisation modeste les personnes de plus de quinze ans désireuses de s'initier à l'informatique. Les enfants peuvent également le faire au sein de l'ADEMIR, affiliée à la fédération. (Cl. Microtel.)

Microtel fête cette année son septième anniversaire. La fédération Microtel (de Micro-informatique et Télécommunication) a été créée le 7 février 1978, à l'initiative d'un petit groupe de passionnés appartenant au C.N.E.T. (Centre national d'études des télécommunications). Le club d'origine est rapidement devenu fédération nationale possédant même des antennes hors de nos frontières. Cette saga est à rapprocher de la création et du développement de la société S.M.T.-Goupil (Société de micro-informatique et télécommunication), également issue d'un groupe d'ingénieurs du C.N.E.T. En sept ans, les temps héroïques ont fait place à une reconnaissance implicite par l'État du rôle d'animateur de Microtel en matière de sensibilisation du grand public à la micro-informatique.

Il s'agissait au départ d'un lieu de rencontre, d'un club pour initiés (à Issy-les-Moulineaux), d'un atelier où les « fanatiques » du fer à souder s'essayaient sur du matériel hétéroclite et primitif. La Fédération nationale Microtel, régie par la loi sur les associations qui s'ensuivit, naquit en 1981. Les subventions proviennent de l'A.D.I. (Agence de l'informatique) et de la D.G.T. (Direction générale des télécommunications). Dès 1979, Microtel donna naissance à l'ADEMIR (Association pour le développement dans l'enseignement de la micro-informatique et des réseaux). Cette création arrivait à point. Elle soulageait les clubs Microtel de l'afflux des jeunes générations qui se manifestaient de plus en plus. L'ADEMIR allait se lier avec le ministère de l'Éducation nationale, Microtel et les municipalités, en prenant comme objectif le développement du sens de l'informatique chez l'enfant. Elle devait rencontrer un vif succès avec quatre-vingt-dix clubs initiant deux mille personnes chaque année scolaire. Depuis 1979, elle répand l'utilisation de ces techniques nouvelles en milieu scolaire. L'idée de l'ordinateur, outil d'E.A.O. (enseignement assisté par ordinateur), faisait alors son chemin avec la participation d'enseignants à l'écriture des programmes, mais aussi avec celle des élèves, incités par là à la créativité.

La formation des jeunes à l'informatique s'ouvre ainsi de manière concrète sur une préparation technique à la vie professionnelle. La force de cette présence associative sur le terrain scolaire réside dans le bénévolat des enseignants, initiés à l'informatique par les stages de la fédé-

ration. Cet esprit d'échange et d'initiative constitue la manière d'être, le « style » Microtel.

Laurent Virol, président de Microtel, définit ainsi la raison d'être de Microtel : « Une présence technique locale, une animation relevant de la vie associative et une dimension potentiellement professionnelle. » Microtel compte aujourd'hui deux cent soixante clubs affiliés. Un club, c'est un local, du matériel et des bénévoles, plus un projet. Le club doit ensuite être reconnu au niveau national. Tous les clubs (dont les quatre-vingt-dix clubs ADEMIR) bénéficient du soutien de la fédération, c'est-à-dire des rencontres inter-clubs, colloques et séminaires ; de la diffusion de bulletins ; de la programmathèque ; de la liaison télématique par Minitel via le réseau Microdial.

Le succès de Microtel est dû à sa situation de pionnier, à une époque où il n'existait aucune structure d'accueil sur le terrain. Depuis, et avec l'évolution de la micro-informatique, le Centre mondial a été créé, et, tout dernièrement, les centres X2000 dans le cadre du projet « Informatique pour tous ». Ces derniers sont dotés de moyens énormes par rapport à ceux de Microtel. Ils ont donc eu tendance dans un passé récent à être vécus comme concurrents. De son côté, la mission Trigano — Nouvelle Formation — représente une évolution par rapport aux centres X2000, dans la mesure où l'informatisation des écoles devrait associer les centres ADEMIR. Il reste que cette reconnaissance par l'État du rôle de l'informatique a été l'occasion pour Microtel de redéfinir ses orientations.

Une pratique aussi décentralisée de la micro-informatique que celle de Microtel n'a pas manqué de susciter des conflits et des convoitises. Il y a eu l'affaire Tandy, dans laquelle Microtel était accusé de piratage de logiciels. Les clubs ont la réputation (justifiée) de bricoleurs. Il est vrai que le côté « petit génie » de Microtel a de quoi inquiéter les fabricants de matériels et les éditeurs de logiciels. Ils peuvent se sentir dépossédés de leurs produits. Les débuts de la micro-informatique sont jalonnés d'histoires de logiciels « déplombés », de matériels « bidouillés ». C'est là que réside la créativité de Microtel. Mais les temps héroïques touchent à leur fin ; le marché se referme sur des standards de plus en plus impératifs.

Les implications de Microtel dépassent largement la sensibilisation à l'informatique. C'est un peu le paradoxe de cette association à but non lucratif : produire des logiciels et diffuser des idées ne devant pas être commercialisées. Cette situation permet néanmoins à Microtel de négocier avec Tandy, Hachette (Alice), Nathan et Hatier. Comme le dit Laurent Virol, il s'agit de « créer de l'humus ». C'est ainsi que se font souvent des accords de troc.

Les clubs Microtel ne font donc pas que réaliser des gestions informatisées de concours de pêche (Club de Vanves, août 1984), ou le suivi

Photo des temps « héroïques » de Microtel, où le fer à souder intervenait fréquemment sur les cartes des non moins pionniers Goupil 2. Illustre le côté « hard » de l'activité ingénieuse des clubs. (Cl. Microtel.)

Longtemps laissé de côté, l'aspect télématique va probablement prendre un nouvel essor avec l'engouement pour le Minitel, l'annuaire Microtel étant accessible. La Bretagne, où le Minitel est bien implanté, a de nombreux clubs Microtel serveurs pour leur environnement. (Cl. Microtel.)

du Tour de France (Saint-Léonard-de-Noblat, été 1985), ce projet faisant intervenir une consultation par Minitel, et un microserveur, avec une carte d'interface sur G3; ils collaborent avec diverses instances de la vie économique et sociale : DASS, Chambres de commerce, municipalités, etc.

De même que l'ADEMIR développe des didacticiels et permet la pratique de l'E.A.O., de même d'autres associations en liaison avec Microtel regroupent des utilisateurs d'une même profession : l'A.M.I.I. (Association médicale d'informatique individuelle), le CIPA (Club d'informatique pour la profession d'avocat et avoué) et AMIPOSTE (Association pour la micro-informatique et la télématique à la poste).

Comment se positionne Microtel par rapport au Centre mondial et à ses antennes, les centres X2000, et vis-à-vis de l'informatisation de l'école? Pour son directeur, le Centre mondial n'occupe pas le même terrain; il est plutôt orienté vers la recherche et les contacts internationaux. En revanche, les centres X2000 posent un problème, reconnaît-il. Microtel ne peut rivaliser en moyens mais estime pouvoir leur être complémentaire. La fédération Microtel mise sur une « certaine économie de ressources » et sur la motivation de ses adhérents, leur côté « militant ». L'accent est mis sur une bonne gestion des clubs, leur efficacité (ils doivent chaque année avoir des réalisations à leur actif). Microtel favorise la créativité par une vie associative éprouvée et peut prétendre se situer en dehors des contraintes commerciales (matériel « confié », logiciels « en test »). Elle peut également compter sur une utilisation optimale du matériel par ses recherches technologiques.

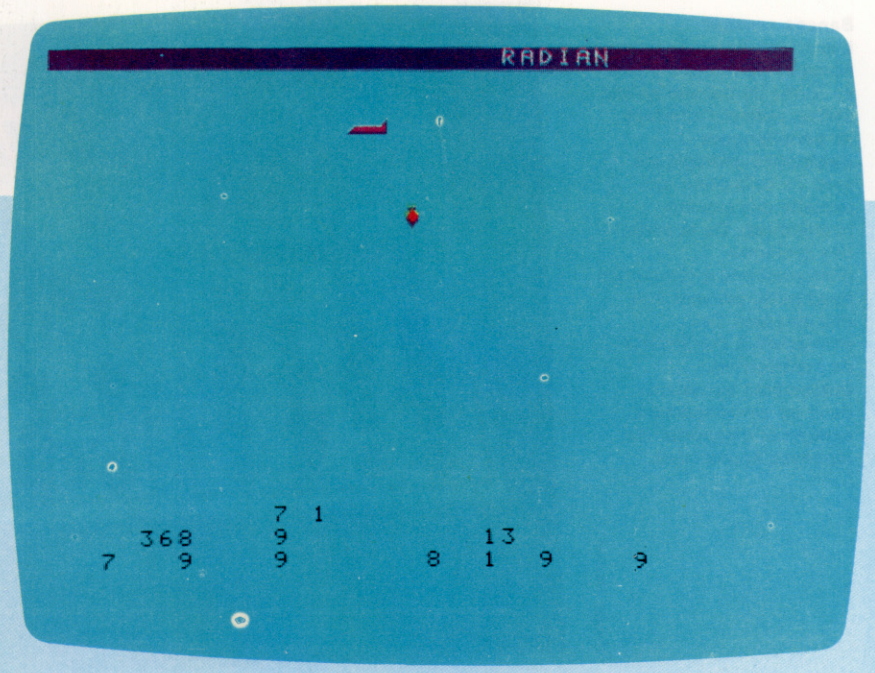
Microtel peut donc apporter un savoir-faire et une expérience aux adhérents des centres X2000. Pour ce qui est de l'informatique à l'école, il est prévu une ouverture des écoles vers l'extérieur, l'ADEMIR pouvant intervenir dans ce cadre : 50 000 écoles et 100 000 micros, telle est l'ampleur de l'informatisation de l'enseignement. ADEMIR peut jouer à cet égard un rôle d'« accélérateur », dans la mesure même de l'immense besoin en formateurs. En effet, avec, à Pâques, 10 000 stages de formation destinés aux enseignants, et 100 000 cet été, l'ambition est simplement d'apprendre aux élèves à utiliser le matériel, et de mettre à leur disposition une « valise de logiciels ».

L'évolution des logiciels inquiète également Microtel. Ils sont de plus en plus ardus, d'accès difficile. La fédération accepte de se les procurer au prix coûtant mais souhaite passer des accords avec les fabricants pour disposer de logiciels de démonstration, établir une sorte de banc d'essai. Il importe qu'une réelle prise de conscience de la valeur des logiciels ait lieu de part et d'autre. Et, comme le dit Laurent Virol, il s'agit d'un « moyen pratique de test, au double sens anglais *test/taste* (essai/appréciation) ».

Numerix sur EXL 100

Pour les possesseurs du micro-ordinateur EXL 100, voici le programme d'un jeu écrit par Pierre Monsaut. Dans le style « des chiffres et des bombes... ».

Dans ce jeu, vous devez bombarder à l'aide de votre avion les chiffres qui se trouvent en bas de l'écran afin d'ajouter ceux-ci à votre total de points. Pour lâcher une bombe, tapez une touche quelconque. Chaque chiffre atteint augmente le nombre de bombes disponibles.

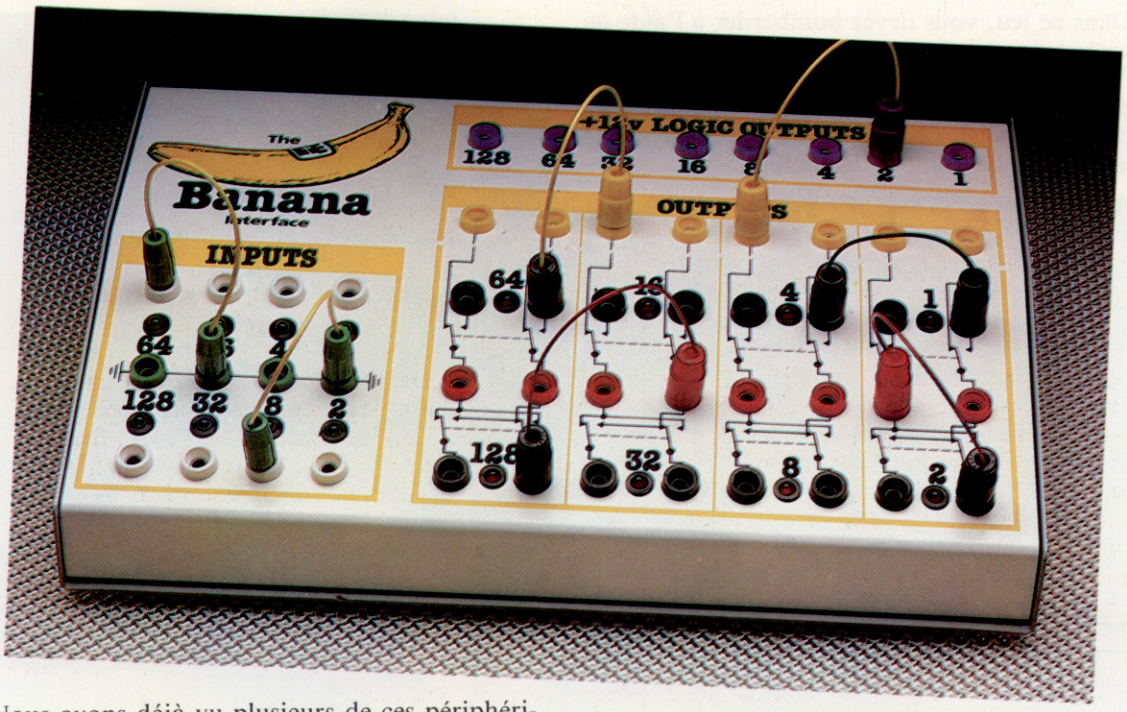



```
100 REM ***********
110 REM * NUMERIX *
120 REM ***********
130 DIM TB(40,22)
140 R=0
150 GOSUB 850
160 GOSUB 560
170 LOCATE (AY,AX)
180 CALL COLOR("1MC")
190 PRINT A$;
200 CALL COLOR("1RC")
210 CALL KEY1(D3,D4)
220 IF D3<>255 AND BY=0 THEN BX=AX:BY=AY
+1:NM=NM-1
230 IF BY<>0 THEN BY=BY+1
240 IF BY>22 THEN LOCATE (BY-1,BX):PRINT
 N$;:BY=0:IF NM<1 THEN 410
250 IF BY<>0 AND TB(BX,BY)<>0 THEN GOSUB
 310
260 IF BY<>0 THEN LOCATE (BY-1,BX):PRINT
 N$;:LOCATE (BY,BX):PRINT B$;
270 IF BY=0 THEN FOR I=1 TO 5:NEXT I
280 AX=AX-1
290 IF AX<1 THEN AX=38:LOCATE (AY,1):PRI
NT M$;
300 GOTO 170
310 LOCATE (BY-1,BX)
320 PRINT N$;
330 LOCATE (BY,BX)
340 PRINT N$;
350 S=S+TB(BX,BY)*10
360 TB(BX,BY)=0
370 BY=0
380 NM=NM+.5
390 GOSUB 720
400 RETURN
410 CALL COLOR("1BC")
420 LOCATE (10,15)
430 PRINT "SCORE :";S;
440 IF S>R THEN LET R=S
450 LOCATE (13,15)
460 PRINT "RECORD :";R;
470 LOCATE (16,15)
480 PRINT "UNE AUTRE ?";
490 CALL KEY1(D3,D4)
500 IF D3<>255 THEN 490
510 CALL KEY1(D3,D4)
520 IF D3=255 THEN 510
530 IF D3<>78 THEN 150
540 CLS
550 END
560 CLS ("BCC")
570 N$=CHR$(32)
580 A$=CHR$(100)&CHR$(101)&N$
590 AX=38
600 AY=3
610 B$=CHR$(102)
620 M$=N$&N$&N$
630 BX=0
640 BY=0
650 GOSUB 810
660 FOR I=1 TO 15
670 GOSUB 720
680 NEXT I
690 NM=20
700 S=0
710 RETURN
720 J=INTRND(9)
730 X=INTRND(37)+1
740 Y=INTRND(3)+19
750 IF TB(X,Y)<>0 THEN 730
760 CALL COLOR("1BC")
770 LOCATE (Y,X)
780 PRINT CHR$(J+48);
790 TB(X,Y)=J
800 RETURN
810 CALL CHAR(100,"0000000000003F7FFF00"
)
820 CALL CHAR(101,"000000000103FFFFFF00"
)
830 CALL CHAR(102,"002810387C7C7C381000"
)
840 RETURN
850 FOR I=1 TO 40
860 FOR J=18 TO 22
870 TB(I,J)=0
880 NEXT J
890 NEXT I
900 RETURN
```

© SYBEX, 1985

Bananarama

L'interface Banana, produite par Castle Associates, destinée principalement au marché de l'éducation, est un périphérique peut-être un peu limité pour une utilisation sur micro domestique.



Crispin Thomas

Banane sensible
L'interface Banana, ainsi nommée du fait des prises en forme de banane qu'elle utilise, permet l'interface de périphériques électriques analogiques avec un ordinateur (qui exploite des signaux numériques). De la sorte, des systèmes d'informations à rétroaction *(feed back)* peuvent être mis en place. Ils permettent à l'ordinateur de contrôler les événements survenant dans un environnement externe et d'apporter les réponses appropriées.

Nous avons déjà vu plusieurs de ces périphériques qui permettent aux ordinateurs de contrôler des mécanismes externes. Parmi les derniers modèles du marché, l'interface Banana est principalement destinée au milieu scolaire. Elle permet le contrôle de divers projets par le Commodore 64, en particulier.

De manière schématique, le concept à la base de l'interface Banana est le même que celui de la boîte-tampon que nous avons déjà construite, même s'il est plus complexe. Des signaux de 5 V sont transmis à des périphériques externes en manipulant les suites de bits contenues dans les données et dans les registres de directions pour les données. En outre, comme les ports utilisateur et imprimante sont bidirectionnels, l'ordinateur peut recevoir des signaux en provenance du périphérique. Il dispose donc d'informations sur l'état du périphérique, par exemple sa position ou, encore, des informations relatives au fait que certaines commutations clés ont été activées. Ainsi, l'ordinateur peut guider avec précision un robot par une combinaison de signaux en entrée et en sortie. Il est donc susceptible de réagir aux circonstances changeantes de l'environnement du robot.

L'interface se connecte à l'ordinateur par un câble-ruban destiné aux ports imprimante et utilisateur. L'autre extrémité du câble est reliée à un connecteur 40 pistes branché dans l'interface Banana. Son alimentation est externe et double. La première, de 5 V, est fournie par l'ordinateur via le câble-ruban. Elle est utilisée pour les circuits logiques des lignes de données de l'interface.

La deuxième source d'alimentation est externe. Elle est transmise par de petites fiches à l'arrière et à côté de l'interface du câble-ruban. Le fabricant, Castle Associates, n'a malheureusement pas prévu d'alimentation 12 V pour l'interface, parce qu'il est destiné avant tout aux établissements scolaires, abondamment pourvus en alimentations de tous types dans leurs laboratoires et ateliers. Aussi, le fait que ces derniers n'aient pas besoin d'alimentations externes a incité les constructeurs de l'interface à ne pas en prévoir, faisant ainsi baisser le prix du matériel.

Cela signifie que les particuliers désirant acquérir cette interface devront se procurer par eux-mêmes une alimentation. Comme les alimentations 12 V de faible ampérage sont relativement rares, il faudra probablement en bricoler une en branchant plusieurs batteries en série (le constructeur recommande un ampérage de 600 mA, bien que la machine que nous avons utilisée fonctionne sans problème sur une alimentation de 10 V, 1 A).

Castle Associates consulta, pour la conception de l'interface Banana, des professeurs de technologie et des designers. Le résultat fut très

positif pour les caractéristiques du matériel, notamment pour ce qui intéresse le monde de l'éducation. Avec la totalité du boîtier en acier, cette interface est l'un des périphériques les plus solides disponibles pour micro-ordinateur.

Cette préoccupation de robustesse s'est également étendue aux composants internes. La carte à circuits imprimés est soudée aux prises de l'interface, maintenant la fragile circuiterie bien en place. Le résultat est un périphérique pouvant probablement supporter toutes sortes de mauvais traitements; il faudrait un élève particulièrement décidé à tout casser pour l'endommager sérieusement...

Le dessus du boîtier comporte plusieurs rangées de miniconnecteurs divisés en trois groupes. Sur la gauche, huit lignes blanches en entrée. Entre elles, quatre lignes de terre, vertes. Chaque connecteur reçoit un numéro selon la progression 1, 2, 4, 8, etc. Le numéro de connecteur correspond au nombre contenu dans chaque position de bit du registre. A côté des positions de bits, les diodes électroluminescentes indiquent les bits de poids fort (ils signalent qu'un courant de 5 V passe dans la ligne de données correspondante).

L'ordinateur peut enregistrer des modifications dans les lignes de données. La meilleure façon de le montrer est de mettre toutes les lignes en position haute. Soumettre le registre à PEEK donne la valeur 255, mais si nous connectons un fil à une des lignes de terre, d'une part, et à une des prises blanches, d'autre part, la tension tombe à zéro. La valeur du registre tombe alors en conséquence, selon la ligne utilisée. On peut construire de la sorte des circuits d'interruption ou circuits d'alarmes, pour des applications d'alarmes antivol par exemple.

Les lignes de sortie figurent à droite des lignes d'entrée. Elles occupent presque toute la place. Comme pour les lignes d'entrée, il y a huit positions de base (numérotées 1, 2, 4, 8, et ainsi de suite jusqu'à 128), des diodes correspondantes pour chaque position de bit, et quatre supports de miniconnecteurs. Les liaisons entre fiches connecteurs se font par circuits-relais. Mettre un nombre dans un registre par POKE à l'adresse &FE61 active les relais.

En connectant les moteurs électriques et les alimentations appropriées des terminaux, ces relais peuvent être utilisés pour contrôler les moteurs et les positionner. Il est possible de diriger de la sorte jusqu'à quatre moteurs électriques par l'intermédiaire de l'interface Banana. Manifestement, la meilleure manière de contrôler ces moteurs (susceptibles d'être connectés à une tortue de sol ou à un bras de robot) est de les piloter par l'intermédiaire des commutateurs de sortie et de diriger leurs mouvements via les ports d'entrée.

Il y a également, sur le dessus de l'interface, un jeu de huit ports logiques haute vitesse. Chacun produit un courant de 12 V. Ces lignes de sortie servent à la conduite de moteurs à impulsions depuis l'interface. Sept moteurs de ce type peuvent être simultanément pris en charge.

L'interface Banana semble avoir eu du succès en Grande-Bretagne auprès des instituts de technologie et de recherche.

L'intérêt d'un tel périphérique (trop coûteux) est moins évident pour l'utilisateur d'un ordinateur domestique.

INTERFACE BANANA

DIMENSIONS

300 × 200 × 63 mm.

INTERFACES

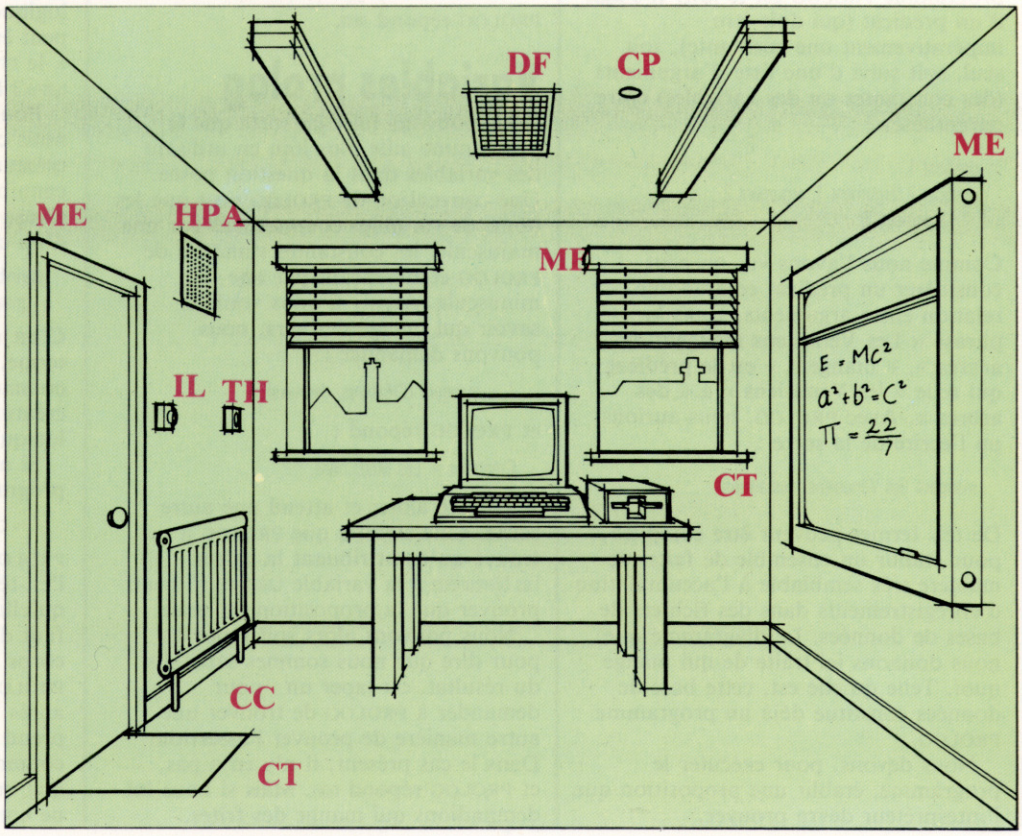
Connecteur parallèle 40 pistes depuis les ports imprimante et utilisateur du Commodore 64. Ports de sortie, et huit prises de sortie 12 V.

AVANTAGES

L'interface Banana est robuste, bien construite, et permet de nombreuses applications de contrôle de périphériques.

INCONVÉNIENTS

L'absence d'alimentation 12 V appropriée limite son marché potentiel aux amateurs spécialisés et aux professionnels : enseignement, technologie et recherche.



Kevin Jones

Sous votre peau
L'interface Banana peut être utilisée pour toutes sortes d'applications de contrôle de périphériques, tant par un amateur que par un professionnel. Par exemple, l'environnement de la pièce représentée ci-contre peut être placé presque entièrement sous le contrôle d'un ordinateur connecté à cette interface. Un certain nombre de périphériques-capteurs, tels que thermostats et cellules photo-électriques, sont placés dans la pièce aux endroits appropriés. Les informations qu'ils transmettent sont analysées par l'ordinateur et comparées à des échelles de valeurs-tests pour optimisation. Toutes les modifications à apporter à l'environnement de la pièce (augmenter la température, baisser les stores, allumer les lumières le soir...) peuvent alors être transmises par les lignes de sortie de l'interface Banana.

CT capteur tactile
CC contrôle chaleur
IL interrupteur lumineux
ME moteur électrique
HPA haut-parleur alarme
DF détecteur de fumée
TH thermostat
CP cellule photoélectrique

Déclarer son intérêt

PROLOG résout les problèmes en ayant recours à une base de données de structures appelées « faits », et en appliquant ces derniers aux propositions par essais successifs. Étudions donc la nature déclarative du langage.

PROLOG est conçu pour donner au programmeur des moyens de décrire la structure logique d'un problème à l'aide d'une bibliothèque de « faits » et de règles « SI...ALORS... ». Cet énoncé déclaratif du problème est utilisé par les mécanismes déductifs de PROLOG pour trouver des réponses aux questions que vous avez posées. La tâche du programmeur n'est donc plus de dire à l'ordinateur, avec de fastidieux détails, comment utiliser les opérations d'un langage (pour résoudre un ensemble prédéfini de problèmes), mais de fournir un énoncé clair et logique des éléments du problème, et de laisser PROLOG faire le reste.

La syntaxe de PROLOG est très simple, même si sa terminologie est assez obscure. L'élément syntaxique le plus important en est le *terme*. Un terme peut être une constante (telle que « arbre », « Jean » ou « 25 »), une variable ou une structure. Les structures sont obtenues à partir de constantes et de variables. Une des structures les plus utilisées dans un programme PROLOG est le *fait*. Il s'agit d'un prédicat (qui doit être impérativement une constante), soit seul, soit suivi d'une liste d'arguments (des constantes ou des variables) entre parenthèses :

```
prédicat 1.
prédicat 2 (argument 1, argument 2,
    argument 3).
```

Comme nous l'avons vu, on peut considérer un prédicat comme une relation entre arguments. Dans la phrase « Les Vénusiens mangent des arbres », « mangent » est le prédicat qui relie « les Vénusiens » à « des arbres ». Avec PROLOG, nous aurions pu l'écrire de la sorte :

```
mangent (les Vénusiens, des arbres).
```

De tels termes peuvent être compilés pour établir un ensemble de faits, de manière très semblable à l'accumulation d'enregistrements dans des fichiers de bases de données. Le diagramme que nous donnons ici traite de qui mange quoi. Telle qu'elle est, cette base de données constitue déjà un programme PROLOG.

Nous devons, pour exécuter le programme, établir une proposition que l'interpréteur devra prouver. Supposons que nous voulions savoir si les Jupitériens mangent des rochers. Nous posons cette question ainsi :

```
? – mangent (Les Jupitériens, des rochers).
```

après le message guide-opérateur du système (habituellement ? –). PROLOG passe en revue les faits proposés, et s'il trouve une proposition correspondant à la question, il répond « oui », ce qui signifie « Oui, le programme montre que mangent (Les Jupitériens, des rochers) est vrai ». Si nous demandons ensuite :

```
? – mangent (Les Martiens, des frites).
```

PROLOG répond non.

Qui mange quoi

```
mangent(Les Vénusiens, des arbres).
mangent(Les Jupitériens, des rochers).
mangent(les Terriens, des frites).
mangent(les Mercuriens, des vers).
mangent(les Neptuniens, des frites).
mangent(Les Martiens, Les Martiens).
```

Les « faits » PROLOG sont exprimés sous la forme d'un prédicat (ici « mangent »), soit seul, soit suivi d'une liste d'arguments. Les faits sont utilisables en tant que base de données de connaissances, susceptible d'être interrogée par l'utilisateur.

Variables prolog

Nous pouvons faire en sorte que le programme aille plus loin en utilisant des variables dans la question posée. Une convention de PROLOG veut que les noms de variables commencent par une majuscule, les constantes standard de PROLOG commençant par une minuscule. Aussi, si nous voulons savoir qui mange des vers, nous pouvons demander :

```
? – mangent (Créatures, des vers).
```

et PROLOG répond :

```
Créatures = Les Mercuriens.
```

Il s'arrête alors, et attend une autre saisie. Cela signifie que PROLOG a trouvé qu'en attribuant la valeur Les Mercuriens à la variable Créatures, il peut prouver que la proposition est vraie.

Nous pouvons alors soit taper RC, pour dire que nous sommes satisfaits du résultat, ou taper un ; pour demander à PROLOG de trouver une autre manière de prouver l'assertion. Dans le cas présent, il n'y en a pas, et PROLOG répond non. Mais si nous lui demandions qui mange des frites, il donnerait deux réponses : les Terriens et les Neptuniens. Aussi en réponse à :

```
? – mangent (Créatures, des frites).
```

PROLOG dit :

```
Créatures = Les Neptuniens
```

S'il y avait d'autres solutions, la frappe d'un ; après chaque réponse permettrait de toutes les connaître. Passons maintenant à trois concepts fondamentaux de PROLOG : les règles, les chaînes d'inférence et le fait de remonter dans un programme.

Une clause est constituée d'un ou plusieurs termes, susceptibles d'exprimer une règle. Une clause a toujours un en-tête constitué d'un terme. Vient ensuite éventuellement le corps de la clause avec un ou plusieurs termes. L'en-tête et le corps sont séparés par le symbole-opérateur :–, qu'on lit normalement « si ». Ainsi nous obtenons :

```
terme 1 :– terme 2, terme 3, terme 4.
```

Les virgules entre les termes du corps peuvent être lues comme des ET logiques. Aussi cet exemple de structure peut être interprété comme signifiant : « le terme 1 est valide SI le terme 2 ET le terme 3 ET le terme 4 sont tous valides ».

Pour plus de clarté, supposons que nous cherchions à savoir si des créatures de notre base de données sont cannibales. Nous pouvons écrire une clause qui nous donne une définition d'un cannibale :

```
cannibale (Créatures) :–
    mangent (Créatures, Créatures).
```

C'est une clause dont le corps a un seul terme. Nous pouvons la comprendre comme une règle qui dirait : des créatures sont des cannibales lorsqu'elles mangent leurs semblables.

Si nous ajoutons cette règle à notre programme et si nous demandons :

```
? – cannibale (X).
```

PROLOG compare cette proposition à l'en-tête de notre nouvelle clause. Parce qu'elle n'est pas nécessairement vraie, il faut d'abord prouver que les termes du corps de la clause sont vrais. Aussi, PROLOG considère chaque terme l'un après l'autre, de gauche à droite. Il les prend pour autant de propositions, comme s'ils avaient été tapés en tant que questions. Dans ce cas, le corps ne comporte qu'un seul terme :

..., mangent (Créatures, Créatures).

ce qui correspond au fait :

mangent (Les Martiens, Les Martiens). Cela établit que Créatures = Les Martiens et signifie que PROLOG Cannibales (Les Martiens) peut être prouvé. Cela établit à son tour X = Les Martiens (X est le nom de la variable dans la question originelle). PROLOG répond :

X = Les Martiens

Progression

Dans ce cas, mangent (Les Martiens, Les Martiens) a été trouvé en tant que fait dans la base de données. Si cela n'avait pas été un simple fait mais une autre règle, PROLOG aurait dû établir de lui-même de nouvelles sous-propositions pour essayer de prouver la véracité de l'en-tête de la règle.

Le programme PROLOG du diagramme ci-contre montre comment cela peut se faire. Si nous posons la question :

? – couleur_de (martien, Couleur).

pour trouver la couleur de martien, PROLOG considère d'abord notre recherche comme sa proposition de départ. Il trouve ensuite une règle pour savoir si couleur_de (martien, rose) est vrai et découvre que c'est bien le cas si (:–) l'humeur de martien est heureuse. En faisant de cela sa proposition suivante, PROLOG trouve alors une règle qui dit martien est heureux s'il peut effectivement programmer avec PROLOG. Ainsi, peuvent_programmer_avec (martien, Prolog) devient la prochaine proposition.

Pourquoi les Martiens sont-ils bleus?

```
Couleur_du(Martien, rose) :– humeur_du
(Martien, heureux)

couleur_du(Martien, bleu)

humeur_du(Martien, heureux) :–
peut_programmer_en(Martien, prolog).

humeur_du(Martien, triste).

peut_programmer_en (martien, basic).

peut_programmer_en (lunien, prolog).
```

Remarquez dans cet ensemble de termes que les noms de variables commencent par des lettres majuscules. Le signe := peut être interprété comme un « SI » logique.

Cette arborescence pourrait se poursuivre sur un grand nombre de niveaux. Mais, en l'occurrence, elle s'arrête là à cause de l'absence de règle qui prouve ou indique comment prouver que martien est un programmeur en PROLOG.

PROLOG ne se contente pas alors d'abandonner ; il reconnaît son impuissance à conclure sur la proposition en cours et remonte à la proposition antérieure. Il tente alors de confirmer d'une autre manière cette proposition. C'est un nouvel échec dans le cas présent (on ne peut prouver que l'humeur de martien est d'être heureux). Aussi PROLOG remonte à nouveau dans le programme pour voir s'il n'existe pas un autre moyen de prouver la toute première proposition. Et il en existe bien un : PROLOG découvre le fait couleur_de (Martiens, bleu). Il établit la relation Les Martiens = martien (martien est adjectif), et la proposition est avérée.

Cette progression le long d'enchaînements de règles et cette façon de poser à chaque nouvelle étape une nouvelle proposition à démontrer constituent l'approche de PROLOG face à toutes les recherches. La méthode s'appelle « remonter un programme ». En effet, lorsqu'une voie n'aboutit pas (pas de réponse), il revient en arrière sur un embranchement antérieur pour reprendre sa progression dans une nouvelle direction. De la sorte, soit il trouve la réponse, soit il continue d'explorer toutes les voies d'une règle à l'autre.

C'est pourquoi l'image qui illustre le mieux un programme écrit avec PROLOG est un arbre, ou enchaînement de propositions, et non une liste d'instructions.

Le programme Alvey

Le langage PROLOG tient une place centrale dans le programme de recherche Alvey de développement de super-ordinateurs de cinquième génération.
Ce programme a été lancé en 1983 à la suite du rapport de la commission Alvey. Cette dernière devait définir le champ des recherches. Elle a été constituée en réponse à l'initiative japonaise de projet d'ordinateur de cinquième génération.
Le projet Alvey insiste sur le décloisonnement des recherches, et préconise une vaste collaboration, notamment avec le projet européen Esprit. Des contacts ont été également pris avec les Japonais.
Environ cent projets ont été approuvés, tous faisant partie du projet central relevant d'une stratégie remise à jour chaque année.
Parmi les diverses études faisant intervenir PROLOG, on trouve le développement d'une machine qui lui est consacrée. Ses principes de base pourraient d'ores et déjà être mis en œuvre avec la technologie existante. Son prix final au niveau de la vente au détail serait substantiellement inférieur à celui d'un mini-ordinateur courant. En utilisant les caractéristiques de base de données intelligente de PROLOG, une telle machine pourrait être directement mise en œuvre en tant que système expert.
On peut noter, parmi les autres applications de PROLOG, son utilisation comme partie d'une interface en langage naturel. Il convient particulièrement au traitement de celui-ci. Cela signifie que, compte tenu de l'importance de cet aspect pour des machines réellement conviviales et pour la recherche en informatique de cinquième génération, PROLOG est bien placé pour garder une place prépondérante dans le développement des nouvelles technologies.

Computer Weekly

L'influence des parents
Le professeur Bob Kowalski, souvent appelé le « père de la programmation logique », et dont les idées furent utilisées par A. Colmerauer pour le développement du premier interpréteur PROLOG à l'université de Marseille.

Pertes et profits

Votre voyage dans le Nouveau Monde touche à sa fin. Avant de partir, vous devez décider de participer ou non aux troubles politiques locaux, ce qui pourra vous valoir d'énormes profits — ou de tout perdre...

Voici maintenant la phase finale du jeu : les échanges ont pris fin, les marchandises acquises auprès du chef ont été montées à bord. L'équipage a pris soin de renouveler le stock d'eau et de nourriture, et le départ est proche. Vous avez réussi à tout négocier, armes exceptées, et les épices, les statues et les perles devraient vous valoir d'importants profits lors de votre retour au port.

Il se passe alors quelque chose d'inattendu. Au cours de la nuit, un rival du chef se glisse furtivement sur votre bateau et cherche à acheter vos fusils, afin de pouvoir renverser son adversaire. Pour chaque arme, il offre trente perles. Si vous acceptez, vous réaliserez un énorme profit, mais il faut tenir compte de plusieurs facteurs.

Si le chef découvre que vous avez cédé les fusils, il cherchera à se venger si d'aventure la rébellion échoue. Vous pouvez naturellement estimer que c'est un risque à courir, au cas où votre marge de profit paraîtrait trop faible.

Si vous refusez la proposition qui vous est faite, le chef, pour vous manifester sa reconnaissance, vous offrira des provisions pour le voyage du retour, et y ajoutera cinquante perles. En bref, si vous acceptez de vendre vos armes, la rébellion a des chances de réussir ; mais les choses tourneront très mal dans le cas contraire, et la partie connaîtra une fin sans gloire...

Si vous réussissez à vous tirer d'affaire, le bateau prendra le chemin du retour. L'équipage est en pleine santé, les vents vous sont favorables, et le voyage ne prendra pas plus de huit semaines. Une fois arrivé au port, vous pourrez vendre vos marchandises, payer vos hommes et, avec un peu de chance, faire quelques bénéfices. Si par malheur vous n'aviez pas de quoi payer les salaires, vous pourriez toujours vendre votre bateau. Le dernier relevé de la partie donne un exposé des dépenses et des recettes et vous permet de savoir si votre équipée vous a été profitable ou non.

Le programme principal appelle ligne 893 un sous-programme qui commence ligne 10300, et traite de l'offre faite par le rival du chef : vos fusils contre des perles. Il s'assure d'abord qu'il y a bien des armes à bord, car si ce n'est pas le cas, la démarche du rebelle n'a évidemment plus aucune raison d'être. La ligne 10305 examine le second élément du tableau des provisions, OA(2), qui correspond aux fusils. Si OA(2) est égal à zéro, l'affaire tombe à l'eau et le contrôle repasse au programme principal. Dans le cas inverse, l'insurgé fait sa proposition.

La ligne 10326 vous demandera d'y répondre par oui ou par non, tandis que la ligne 10330 vérifiera que votre réponse est bien correcte. Le programme vous reposera la même question si votre entrée n'est pas conforme.

Si vous acceptez de céder les armes, le sous-programme passera ligne 10400. La ligne 10405 génère un nombre aléatoire compris entre 0 et 1 ; il y a 75 % de chances que l'on se retrouve ligne 10450, qui marque l'échec de la rébellion. Vous pouvez naturellement modifier le nombre en question au cas où les probabilités d'échec vous paraîtraient trop élevées.

Si les insurgés sont écrasés, le chef mettra le feu au navire, après vous avoir dépossédé de ce que vous lui aviez acheté. Le programme prendra fin. Dans le cas contraire, il ira en ligne 10429, qui voit la réussite du soulèvement ; le nouveau chef, reconnaissant, vous offre toutes les provisions dont vous aurez besoin pour votre voyage de retour.

Ajuster les tableaux

L'équation de la ligne 10429 calcule le nombre de perles acquises lors de l'échange avec le rival du chef : le nombre de fusils, OA(2), est multiplié par trente, et le résultat est ensuite ajouté au premier élément du tableau A0() afin d'obtenir le total des perles qui vous appartiennent désormais. Le ligne 10430 supprime les fusils du tableau des provisions. Ce n'est sans doute pas totalement indispensable, mais c'est une bonne habitude de programmation. De surcroît, cela vous permettra d'adapter plus facilement le jeu, au cas où par exemple vous désireriez y intégrer divers incidents survenant lors du voyage de retour.

Si vous déclinez l'offre qui vous est faite, le chef, prévenu, vous offrira des provisions et,

Variantes basic

Spectrum :

Remplacez partout A0() par E(), V1() par B() et V2() par D(), et procédez aux modifications suivantes :

```
10310 CLS:GOSUB 9200
10328 INPUT I$:LET I$=I$(TO 1)
10354 LET I$=INKEY$:IF I$=«»THEN GOTO
  10354
10400 CLS:GOSUB 9200
10501 CLS:GOSUB 9200
10547 LET I$=INKEY$:IF I$=«»THEN GOTO
  10547
```

BBC Micro :

Procédez aux modifications suivantes :

```
10310 CLS:GOSUB 9200
10354 I$=GET$
10400 CLS:GOSUB 9200
10501 CLS:GOSUB 9200
10547 I$=GET$
```

Module treize : la fin du voyage

Addition au programme principal

```
893 GOSUB10300
894 GOSUB10500
```

S/P Rébellion

```
10300 REM Rébellion
10305 IFOA(2)=0THENRETURN
10310 PRINTCHR$(147):GOSUB9200
10315 S$="AU COURS DE LA NUIT UN RIVAL*":GOSUB9
100
10316 S$="DU CHEF VOUS REND VISITE*":GOSUB9
100
10317 PRINT:GOSUB9200
10318 S$="IL VEUT ACHETER VOS FUSILS POUR*":GOSUB910
0
10320 S$="UNE INSURRECTION*":GOSUB9100
10322 PRINT:GOSUB9200
10324 S$="IL OFFRE 30 PERLES PAR FUSIL*":GOSUB9
100
10326 S$="ACCEPTEZ-VOUS ? (O/N)*":GOSUB91
00
10328 INPUTI$:I$=LEFT$(I$,1)
10330 IFI$<>"N"ANDI$<>"O"THEN10328
10332 IFI$="O"THEN10400
10334 PRINT:GOSUB9200
10336 S$="LE CHEF EST AVERTI ET*":GOSUB9100
10338 S$="RECONNAISSANT*":GOSUB9100
10340 S$="VOUS OFFRE DES PROVISIONS*":GOSUB9100
10342 S$="POUR LE RETOUR*":GOSUB9100
10344 GOSUB9200
10345 IF RND(1)<.75 THEN 10350
10346 S$="ET 50 PERLES!*":GOSUB9100
10348 AO(1)=AO(1)+50
10350 PRINT:GOSUB9200
10352 S$=K$:GOSUB9100
10354 GETI$:IFI$=""THEN10354
10359 RETURN
10400 PRINTCHR$(147):GOSUB9200
10405 IFRND(1)<.75THEN10450
10410 S$="L'INSURRECTION L'EMPORTE*":GOSUB9100
10412 PRINT:GOSUB9200
10415 S$="LE NOUVEAU CHEF VOUS OFFRE*":GOSUB91
00
10420 S$="DES PROVISIONS POUR LE RETOUR*":GOSUB91
00
10425 S$="EN EUROPE*":GOSUB9100
10429 AO(1)=AO(1)+ OA(2)*30):REM AJOUTE PERLES
10430 OA(2)=0
10431 GOTO10350
10450 S$="L'INSURRECTION ECHOUE!*:GOSUB9100
10452 PRINT:GOSUB9200
10455 S$="LE VIEUX CHEF, FURIEUX*":GOSUB9
100
10457 S$="BRULE VOTRE NAVIRE ET S'EMPARE*":GOSUB910
0
10458 S$="DE TOUT !*":GOSUB9100
10459 PRINT:GOSUB9200
10460 S$="    FIN DE LA PARTIE !*":GOSUB9100
10462 END
10464 GOTO10462
```

S/P Fin du voyage

```
10500 REM FIN DU VOYAGE
10501 PRINTCHR$(147):GOSUB9200
10505 S$="UN EQUIPAGE EN FORME ET*":GOSUB9100
10507 S$="DES VENTS FAVORABLES*":GOSUB9100
10508 S$="PERMETTENT UN VOYAGE DE HUIT SEMAINES*":GOSUB9100
10512 WW=0
10514 FORT=1TO5
10516 WW=WW+(8*CC(T)*WG(T))
10518 NEXT
10519 PRINT:GOSUB9200
10520 S$="TOTAL DES SALAIRES POUR LE RETOUR=*":GOSUB91
00
10522 PRINTWW:"PIECES D'OR*"
10524 PRINT:GOSUB9200
10526 S$="UNE FOIS RENTRE*":GOSUB9100
10528 S$="VOUS VENDEZ VOS MARCHANDISES*":GOSUB9100
10530 S$="ELLES VALENT MAINTENANT:*":GOSUB9100
10532 PRINT"PERLES - " :V2(1):"PIECES D'OR"
10534 PRINT"STATUES - " :V2(2):"PIECES D'OR"
10536 PRINT"EPICES - " :V2(3):"PIECES D'OR"
10538 PRINT:S$="SOIT UN TOTAL DE :*":GOSUB9100
10540 X=(AO(1)*V2(1))+(AO(2)*V2(2))+(AO(3)*V2(3))
10542 PRINTX:"PIECES D'OR"
10545 PRINT:S$=K$:GOSUB 9100:PRINT
10547 GET I$:IF I$="" THEN 10547
10550 S$="VOUS AVEZ ACTUELLEMENT*":GOSUB9100
10552 PRINTMO+X:"PIECES D'OR"
10555 PRINT:GOSUB9200
10556 S$="LE TOTAL DES SALAIRES POUR TOUT LE VOYAGE EST DE*":
GOSUB9100
10557 PRINTWT+WW:"PIECES D'OR"
10559 PRINT:GOSUB9200
10560 S$="IL RESTE DONC":GOSUB9100
10562 Z=MO+X-WT-WW
10565 PRINTZ:"PIECES D'OR"
10566 PRINT:GOSUB9200
10567 PRINT:S$="VOUS ETES VRAIMENT*":PRINT
10568 IF Z>3200 THEN S$="UN SUPER CAPITALISTE!*":GOSUB9
100
100:END
10569 IF Z>2500 THEN S$="UN MAITRE MARCHAND*":GOSUB9100:
END
10570 IF Z>2000 THEN S$="AU DEBUT DE VOTRE CARRIERE*":GOSU
B9100:END
10571 IF Z>1000 THEN S$="ASSEZ PEU DOUE*":GOSUB9100
10572 S$="CONDAMNE A CHANGER DE METIER...*"
10573 GOSUB9100:END
```

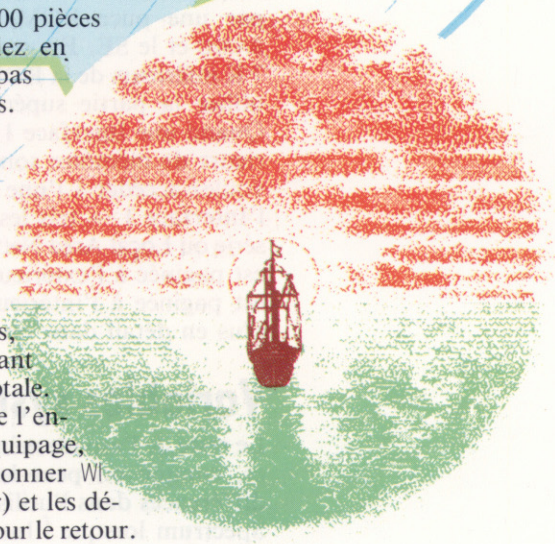
peut-être, cinquante perles, ce qui est déterminé ligne 10345 de façon aléatoire. Le dernier sous-programme commence ligne 10500 et se consacre à la traversée lors du retour, ainsi qu'à la vente des marchandises.

Revenir en Europe demande huit semaines et vous devrez connaître le total des salaires. Pour ce faire, une variable WW est créée ligne 10512. Elle représente les sommes à payer pour le voyage de retour. Sa valeur est calculée par la boucle qui commence ligne 10514. La formule multiplie le compte de chaque catégorie de marins, CC(T), par le salaire hebdomadaire de chacune d'elles, WG(T), de façon à obtenir un chiffre global. Le compteur de boucles, T, assure la répétition du processus pour chaque catégorie, ce qui finit par nous donner un chiffre hebdomadaire total, qu'il suffit de multiplier par huit (nombre de semaines du voyage) : on dispose alors du total des dépenses, et le programme en assure l'impression à l'écran. Les marchandises sont vendues lors de l'arrivée au port. Leur valeur marchande est déterminée par le tableau V2(3), DIMensionné ligne 62 : elle a, en fait, été fixée en début de partie de façon aléatoire. Les lignes 10532 à 10536 du sous-programme Fin-du-Voyage affichent toutes les valeurs contenues dans le tableau, pour chaque catégorie de marchandise.

L'or acquis lors de la vente est calculé ligne 10540, et le résultat conservé en X. La formule mise en œuvre à cette occasion multiplie la quantité de chaque marchandise, telle qu'elle est enregistrée dans le tableau AO(), par la valeur marchande du moment, détenue en V2(). Les sommes que rapportent les perles, les statues et les épices sont additionnées, et le total est affiché à l'écran.

Il se peut que les 2 000 pièces d'or dont vous disposiez en début de partie n'aient pas été entièrement dépensées. Ce qui reste éventuellement a été conservé tout au long du périple par la variable MO, et ce surplus, s'il existe, sera ajouté à l'or que vous rapportera la mise en vente des objets, la ligne 10552 provoquant l'affichage de la somme totale.

La ligne 10557 calcule l'ensemble des gages de l'équipage, en se bornant à additionner WW (salaires du voyage aller) et les dépenses du même ordre pour le retour.



Ian McKinnell

Analyse de Spectrum

Nous allons commencer à explorer les aspects du système d'exploitation du Sinclair Spectrum, en inspectant l'agencement de la table d'implantation en mémoire de l'ordinateur.

Le Sinclair Spectrum n'a pas de ROM système d'exploitation distincte à l'intérieur de l'ordinateur, puisque le SE et l'interpréteur BASIC sont tous deux dans la même puce. De plus, il n'y a eu qu'une seule version de cette ROM et, puisque le Spectrum+ utilise la même puce que le Spectrum, cette situation n'a pas lieu de changer. Le SE est un ensemble compact de routines qui ne prend que quelque 7 K de la puce ROM 16 K du Spectrum.

Malgré le manque de variantes dans le SE et la compacité du codage, le SE Spectrum souffre d'un sérieux inconvénient. Il n'est pas vectorisé, mis à part deux exceptions mineures, et il est donc plus difficile de modifier le mode d'action du SE. Il est toutefois possible de contourner cela, et nous l'aborderons ultérieurement.

Si vous avez suivi les articles précédents, vous êtes déjà familiarisé avec le rôle joué par le SE dans l'ordinateur. En bref, ce rôle peut se résumer par le maniement de toutes les routines concernées par les entrées, sorties, affichages, piles et interruptions, et fournissant généralement une interface entre le matériel et le programme utilisateur ou l'interpréteur BASIC. Nous commencerons notre investigation du SE Spectrum en considérant la table mémoire (mappe) de l'ordinateur (voir figure ci-contre).

Le premier bloc mémoire de 16 K est occupé par une puce ROM contenant l'interpréteur BASIC et le SE. En gros, le SE occupe les 6 ou 7 K inférieurs de la ROM, et l'interpréteur BASIC utilise la partie supérieure de la puce ROM. Quand une Interface 1 est utilisée, les 8 K inférieurs de mémoire sont paginés. Lorsque vous voulez accéder à l'une des facilités offertes par l'Interface 1 — tels les Microdrives, l'interface série ou Local Area Network — la ROM normale est paginée à l'extérieur et la ROM Interface 1 est paginée à l'intérieur. Nous étudierons cela plus en détail dans un prochain numéro.

Variables système

Le reste de l'espace mémoire est pris par la RAM et, comme on peut le voir, il y a de légères différences dans l'utilisation de la RAM sur un Spectrum lorsque l'Interface 1 est utilisée.

En ce qui concerne les sections de la mappe (plan d'implantation), le *fichier visu* contient l'information concernant la forme des images qui doivent être visualisées à l'écran. Les données concernant la couleur, BRIGHT et FLASH, ne sont pas contenues dans ce fichier, mais dans un *fichier attribut,* parce que cette information concerne un caractère sur l'écran de visualisation. Le *tampon imprimante* est une zone de 256 octets de mémoire utilisée par l'imprimante ZX en cas de copie sur papier.

Les *variables système* sont utilisées par le SE et l'interpréteur, de sorte qu'elles peuvent surveiller ce qui se passe dans le système. Nous considérerons les plus utiles de ces variables système dans cette série. Une liste complète se trouve dans le manuel utilisateur du Spectrum. Malheureusement, cette liste ne figure pas dans le manuel du Spectrum+.

C'est là que les différences entre Spectrum avec et sans l'Interface 1 sont les plus manifestes. Évidemment, la ROM Interface 1 requiert une certaine mémoire de travail et des variables système; donc, lorsqu'elle est utilisée, elle génère des variables système supplémentaires — lesquelles peuvent être placées après les variables système normales du Spectrum. Bien sûr, cela porte un coup au reste de la mémoire utilisée, et on fait des ajustements en conséquence.

Les *mappes Microdrive* sont également spécifiques à l'Interface 1 et sont utilisées afin que l'Interface puisse garder trace des zones employées et non employées d'une bande Microdrive.

La zone *données canal* nous fournit une forme limitée de vectorisation pour les instructions LLIST LPRINT INPUT# et PRINT#. Par exemple, nous pouvons écrire un programme en langage machine pour commander une imprimante standard et modifier les données canal afin que, lorsqu'on rencontre une instruction LPRINT ou LLIST, elle soit influencée par notre partie de langage machine, plutôt que les routines interpréteur BASIC usuelles. Nous examinerons cela au prochain numéro. La donnée canal est également utilisée par l'Interface 1.

Nous passons ensuite à la mémoire de travail variables et programme BASIC, utilisée par le système pour stocker le texte d'un programme BASIC et les variables utilisées par ce programme.

Les espaces de travail *edit* et *input* servent au système à EDITer une ligne de programme ou à entrer des données ou des lignes de programme. Une fois terminées l'édition ou l'entrée de texte, les lignes de programme sont stockées dans l'emplacement approprié dans le programme, et ces zones sont libres pour être réutilisées.

Puis nous rencontrons un groupe de piles. La *pile calculateur* est utilisée par l'interpréteur BASIC pour ses opérations arithmétiques. La *pile*

Z80 est utilisée par l'UC à peu près de la même façon que le 6502 utilise la page 1 de sa mémoire — c'est-à-dire pour stocker les adresses de retour pour les sous-programmes en langage machine et les données qui ont été mises sur pile pour stockage temporaire. La pile GOSUB fait aussi partie de la mémoire utilisée par l'interpréteur BASIC. Elle stocke des adresses de retour pour les instructions BASIC GOSUB. Enfin, la zone pour les *graphiques définis par l'utilisateur* contiennent évidemment les données pour les caractères définis par l'utilisateur.

Où mettre le code machine

Il ressort clairement de la mappe mémoire qu'aucun espace n'est réservé pour les programmes langage machine utilisateur. Sur le Spectrum, il y a trois zones possibles où on peut mettre le code. C'est dans une instruction REM à la ligne 1 — nous pouvons trouver son emplacement mémoire à partir de la valeur contenue dans la variable système PROG. En additionnant 5 à cette valeur, on obtient l'adresse du premier caractère dans une REM en ligne 1.

Les autres emplacements sont le tampon imprimante et entre RAMTOP (haut de RAM) et la zone de mémoire utilisée pour le stockage des données graphiques définies par l'utilisateur. Le tableau suivant montre les mérites relatifs de ces méthodes :

Emplacement	Spectrum	Spectrum & Interface 1
REM ligne 1	✓	✗
Tampon imprimante	(✓)*	(✓)*
Au-dessus de RAMTOP	✓	✓

* Ne peut pas être utilisé si une imprimante est employée.

Comme vous voyez, seule la dernière option vaut la peine d'être considérée, puisque les autres peuvent être risquées dans certains cas. Tout ce qui est au-dessus de RAMTOP ne risque pas d'être écrasé par BASIC, et il est assez facile d'y réserver de l'espace. La position exacte de RAMTOP est contenue dans une variable à 2 octets aux adresses 23730 et 23731. La valeur stockée au format Z80 standard — octet lo suivi par octet hi ; afin de vérifier la position en cours de RAMTOP, il vous faut donc exécuter :

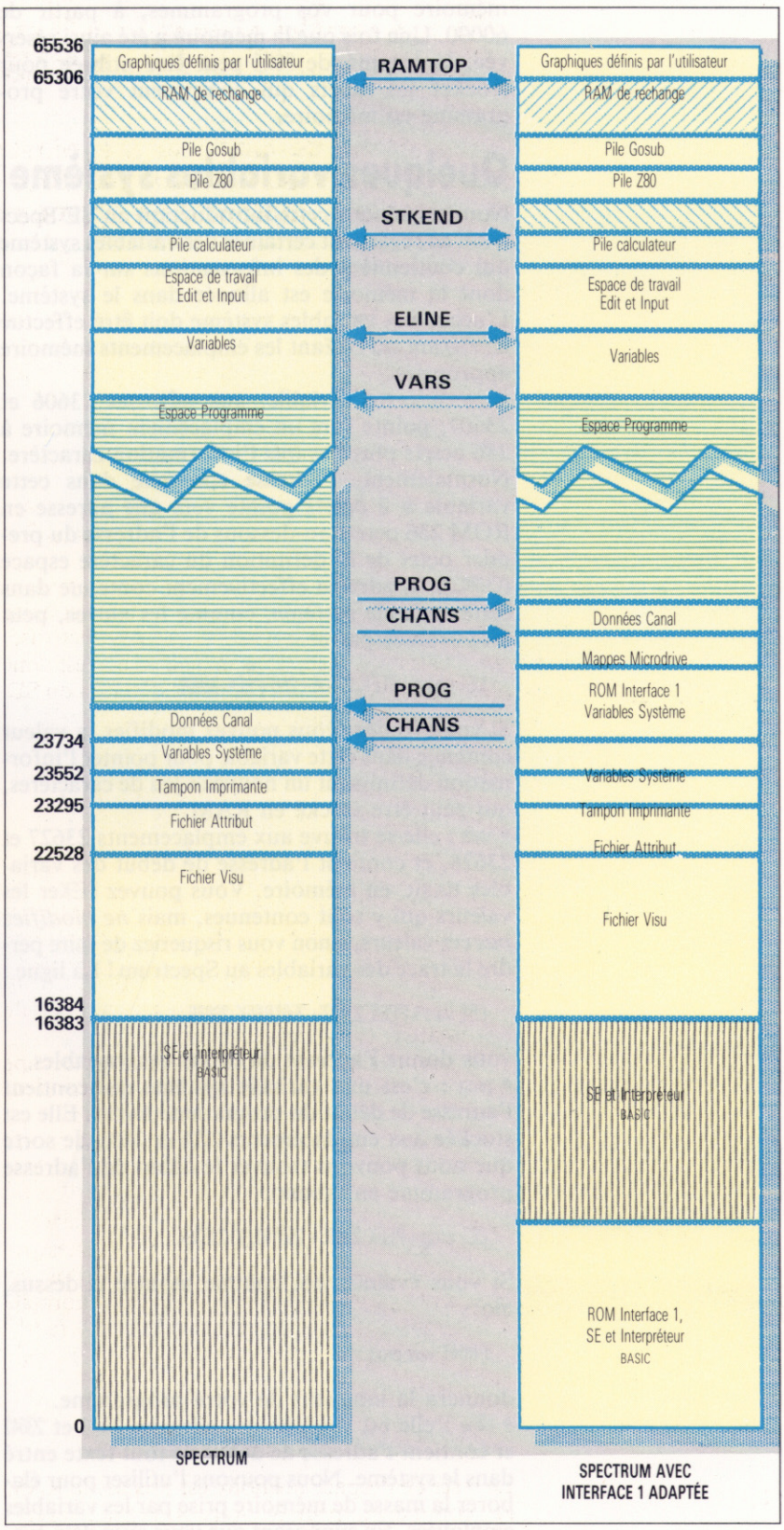
```
LET ramtop=PEEK23730+256*PEEK23731
```

Une fois que vous avez noté la position de RAMTOP (c'est généralement 65306 sur une machine 48 K standard), vous pourrez estimer combien d'octets il vous faudra pour votre programme en langage machine. Ayant trouvé la valeur à l'aide de la formule RAMTOP – (nombre d'octets dans programme + 1), vous pouvez alors abaisser RAMTOP à l'aide de la commande CLEAR :

```
CLEAR value
```

Notez que si vous utilisez cette commande dans un programme BASIC, elle effectuera un CLS (effacement d'écran), réinitialisera la position PLOT, et exécutera un RESTORE. Elle effacera aussi et relogera la pile GOSUB. C'est pourquoi vous devez prendre l'habitude de vous assurer que la commande est bien utilisée et placée chaque fois que possible au début d'un programme — sinon, des problèmes s'ensuivront presque à coup sûr.

Après avoir exécuté la commande CLEAR value, la nouvelle position de RAMTOP sera stockée aux emplacements 23730 et 23731, et le premier octet disponible pour votre programme en langage machine se trouvera à l'adresse (value+1). Ainsi la commande CLEAR 59999 réservera environ 5 K de



Liz Dixon

A l'intérieur des données Notre mappe mémoire pour le Spectrum 48 K (et le Spectrum +) montre comment l'Interface 1 reconfigure certaines zones de RAM Spectrum.

mémoire pour vos programmes, à partir de 60000. Une fois que la mémoire a été ainsi réservée, la commande POKE peut être utilisée pour stocker les octets qui composent votre programme en mémoire.

Quelques variables système

Nous terminons cette introduction au SE Spectrum en regardant certaines des variables système qui contiennent des informations sur la façon dont la mémoire est allouée dans le système. L'accès aux variables système doit être effectué en PEEKant ou POKEant les emplacements mémoire appropriés.

• chars : cette variable, aux adresses 23606 et 23607, pointe vers un emplacement mémoire à 256 octets plus bas que l'information caractère. Normalement, l'adresse contenue dans cette variable à 2 octets pointe vers une adresse en ROM 256 octets au-dessous de l'adresse du premier octet de la définition du caractère espace (CHR$32). L'adresse effectivement contenue dans cette variable système, comme les autres, peut être évaluée par :

```
LET chars=PEEK 23606+256*PEEK 23607
```

Si vous voulez, vous pouvez modifier la valeur contenue dans cette variable pour pointer l'information définissant un nouveau jeu de caractères, qui peut être stocké en RAM.

• vars : elle se trouve aux emplacements 23627 et 23628, et contient l'adresse de début des variables BASIC en mémoire. Vous pouvez PEEKer les valeurs qui y sont contenues, mais *ne modifiez pas* ces valeurs, sinon vous risqueriez de faire perdre la trace des variables au Spectrum ! La ligne :

```
LET vars=PEEK 23627+256*PEEK 23628
```

vous donne l'adresse de début des variables.

• prog : c'est une variable système qui contient l'adresse de début du programme BASIC. Elle est stockée aux emplacements 23635 et 23636, de sorte que nous pouvons trouver le début de l'adresse programme en tapant :

```
LET prog=PEEK 23635+256*PEEK 23636
```

Si vous évaluez vars, comme montré ci-dessus, alors :

```
PRINT vars-prog
```

donnera la longueur de votre programme.

• eline : elle est stockée aux adresses 23641 et 23642 et contient l'adresse de début de tout texte entré dans le système. Nous pouvons l'utiliser pour élaborer la masse de mémoire prise par les variables employées. En supposant que vous ayez déjà évalué vars :

```
LET eline=PEEK 23641+256*PEEK 23642
PRINT eline-vars
```

donnera le total de mémoire pris par vos variables.

• chans : une variable qui contient l'adresse du début d'information canal. Elle se trouve aux adresses 23631 et 23632. Nous considérerons le sujet des canaux plus en détail dans un prochain numéro.

• stkend : elle se trouve aux emplacements 23653 et 23654. La variable, en conjonction avec la valeur de RAMTOP, nous permet d'estimer le total de mémoire laissé pour notre programme et les variables BASIC. Elle est évaluée par :

```
LET stkend+PEEK 23653+256*PEEK 23654
```

Si nous avons déjà évalué stkend, alors le total de mémoire restant est donné par :

```
PRINT ramtop-stkend
```

Nous pouvons également utiliser une routine ROM qui donne une estimation du total de mémoire restant. Cette méthode n'est pourtant pas aussi précise que celle que nous venons d'utiliser. En supposant que nous ayons obtenu la valeur en cours de RAMTOP :

```
PRINT ramtop-USR 7962
```

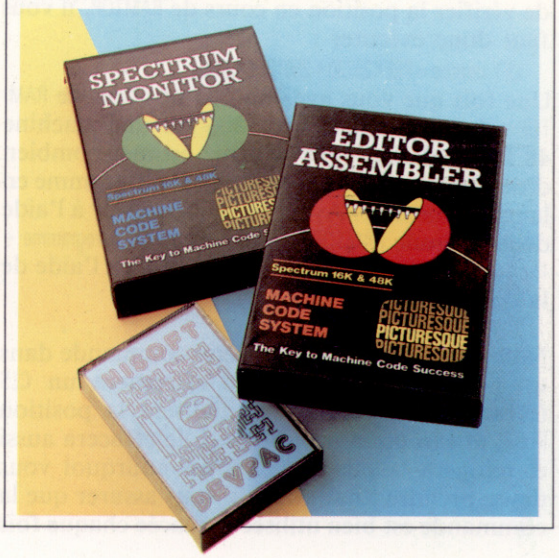
donnera une estimation de l'espace libre.

Temps d'assemblage

Il existe de nombreux logiciels d'assemblage pour l'utilisateur de Sinclair Spectrum intéressé par l'étude du langage machine. L'un des plus populaires est le DEVPAC 3 de Hisoft. Fourni en cassette, il inclut l'assembleur GENS3 et le moniteur-désassembleur MON3. Une version Microdrive du logiciel est aussi disponible. Bien que largement utilisé par les possesseurs de Spectrum, DEVPAC 3 n'est pas l'un des assembleurs les plus agréables à utiliser; toutefois, c'est le moins cher.

Le Picturesque Editor Assembleur et Spectrum Monitor (incluant un désassembleur) est un logiciel double. L'un des principaux avantages du premier est qu'il peut s'adapter sans problème sur un Spectrum 16 K. Les manuels qui l'accompagnent sont supérieurs à la documentation fournie par Hisoft, et tout le système donne une impression de plus grand professionnalisme.

(Cl. Chris Stevens.)



Règle à mesurer

L'un des traits manquant au Spectrum est un moyen convenable pour calculer la longueur d'un programme; cela peut se faire à l'aide de PEEK, mais il existe un utilitaire qui emploie les variables système PROG et VARS pour calculer la longueur de tout programme BASIC en mémoire, et le stocker dans une ligne d'instruction 1 REM. Tout ce qui est nécessaire est que la première ligne de programme soit une instruction REM contenant au moins cinq caractères, comme :

```
1 REM 00000 Bytes Long
```

Lorsque la routine en langage machine est exécutée, l'instruction REM est modifiée pour contenir la longueur du programme.
Par exemple :

```
1 REM 00801 Bytes Long
```

C'est assez utile, puisque la fois suivante où le programme sera examiné, cette longueur sera présente. La première partie de code est un listage d'assemblage du langage machine en question. Si vous n'avez pas un assembleur convenable, le second listage est un programme BASIC qui peut servir à charger le programme en langage machine.
Dans le chargeur BASIC, CLEAR 61999 met la valeur de la variable système RAMTOP à 61999. Le premier octet « sauf » est donc à l'adresse 62000, et nous l'utilisons comme adresse de départ pour la routine en langage machine. RAMTOP restera à cette nouvelle valeur jusqu'à ce qu'il soit modifié par une commande CLEAR n, PRINT USR 0 ou la mise hors tension de la machine. NEW n'affectera pas la valeur de RAMTOP, et la routine en langage machine est protégée contre l'écrasement par BASIC tandis que RAMTOP reste inaltéré.
Une fois le programme en mémoire, on utilise USR pour exécuter le code machine. L'argument de la fonction USR(n) est l'adresse de la routine langage machine que vous voulez exécuter. Cette adresse peut se trouver n'importe où en mémoire. Ainsi, les commandes suivantes exécuteront toutes le code machine à l'adresse 62000, et n'importe laquelle peut être utilisée pour exécuter le programme en langage machine que nous venons d'exécuter :

```
RANDOMISE USR(62000)
PRINT USR(62000)
LET L=USR(62000)
```

Les méthodes PRINT et LET d'utilisation de USR donnent également un résultat utile au BASIC. Le résultat est la valeur qui se trouve dans la paire de registres BC lorsque la routine langage machine se termine et retourne au BASIC; une routine langage machine peut ainsi repasser des résultats au BASIC. Si votre routine n'affecte pas la valeur du registre BC, l'adresse de la routine est donnée.

Listage d'assemblage

```
62000                 10          ORG 62000
62000  2A4B5C         20          LD    HL,(23627)   ;get VARS in HL
62003  ED5B535C       30          LD    DE,(23635)   ;get PROG in DE
62007  AF             40          XOR   A            ;clear carry flag
62008  ED52           50          SBC   HL,DE        ;get program length into HL
62010  DD2A535C       60          LD    IX,(23635)   ;start of program into IX
62014  DD23           70          INC   IX
62016  DD23           80          INC   IX           ;increment IX to point to
62018  DD23           90          INC   IX           ;character after REM token
62020  DD23          100          INC   IX           ;in first program line
62022  DD23          110          INC   IX
62024  011027        120          LD    BC,10000     ;start getting ASCII codes
62027  CD6BF2        130          CALL  PRINT        ;of each digit in program
62030  DD23          140          INC   IX           ;length and store in REM
62032  01E803        150          LD    BC,1000      ;statement
62035  CD6BF2        160          CALL  PRINT
62038  DD23          170          INC   IX
62040  016400        180          LD    BC,100
62043  CD6BF2        190          CALL  PRINT
62046  DD23          200          INC   IX
62048  010A00        210          LD    BC,10
62051  CD6BF2        220          CALL  PRINT
62054  DD23          230          INC   IX
62056  010100        240          LD    BC,1
62059  AF            250  PRINT:  XOR   A            ;clear carry and xero counter
62060  B7            260  L1:     OR    A
62061  ED42          270          SBC   HL,BC        ;subtract power of ten from
                                                     ;program length...
62063  3803          280          JR    C,FINISH     ;and jump if negative
62065  3C            290          INC   A            ;inc A reg if not negative
62066  18F8          300          JR    L1           ;repeat
62068  09            310  FINISH: ADD   HL,BC        ;restore prog length to
                                                     ;positive value for the next
                                                     ;power of ten
62069  C630          320          ADD   A,48         ;get ASCII code of digit
62071  DD7700        330          LD    (IX),A       ;store ASCII code in REM
62074  C9            340          RET                ;finished
```

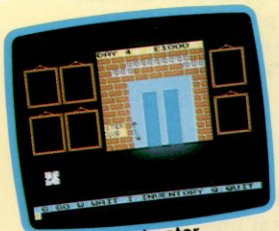
Chargeur basic

```
  10 REM 00000 Bytes Long
  20 CLEAR 61999
  30 FOR i=0 TO 74
  40 READ a: POKE 62000+i,a
  50 NEXT i
1000 DATA 42,75,92,237,91,83,92,175,237,82,221,42,
83,92,221,35,221,35,221,35,221,35,221,35,1,16,39,2
05,107,242,221,35,1,232,3,205,107,242,221,35,1,100
,0
1010 DATA 205,107,242,221,35,1,10,0,205,107,242,22
1,35,1,1,00,175,183,237,66,56,3,60,24,-8,9,198,48,
221,119,0,201
```

Note : pour s'assurer que la règle à mesurer localise la position correcte pour insérer les chiffres indiquant la longueur du programme, ne pas insérer d'espaces entre le premier numéro de ligne et le mot clé REM (autre que l'espace automatiquement inséré par le système de mots clés Spectrum).

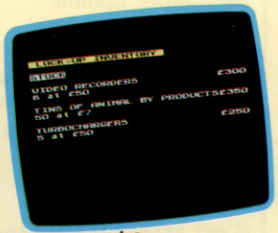
Une bonne affaire

Voici un nouveau jeu produit par Euston Films, dans lequel vous jouez le rôle du fumeur de cigares Arthur, toujours à la recherche d'une bonne affaire et d'un profit rapide.


Le bar Winchester

Les tractations

L'entrepôt


L'inventaire

Une affaire en or
Voici quatre écrans du jeu Minder. L'objectif de ce jeu est de vendre votre marchandise (stockée en entrepôt) à des vendeurs. Si vos marchandises sont épuisées, vous pouvez vous en procurer d'autres en contactant les clients du bar Winchester. Vous devez évidemment marchander pour obtenir le meilleur prix possible lors de l'achat et de la vente, afin de réaliser le profit maximal dans le temps alloué. (Cl. Liz Heaney.)

L'émission télévisée Minder a été l'un des plus grands succès de la télévision britannique au cours des dernières années et sa conversion en jeu arrive seulement maintenant (la série a commencé au milieu des années 1970).

Minder est fondé sur les affaires d'Arthur Daley, un homme d'affaires peu scrupuleux, et sur Terry McCann, son bras droit. Vous jouez dans ce jeu le rôle d'Arthur, et l'objectif est de gagner autant d'argent que possible en vendant et en achetant des marchandises. Vous devez approcher d'autres vendeurs à qui vous pouvez vendre votre marchandise. Vous aurez souvent des choses étranges à vendre, tels des sous-produits animaux ou des appareils de radiographie. Très souvent, vos interlocuteurs vous offriront des prix trop bas et vous devrez marchander pour obtenir un prix raisonnable.

L'un des gros problèmes auquel Arthur doit faire face est la présence du sergent détective Chisholm. Le rôle de Chisholm commence à se faire sentir lorsque vous rendez visite à un vendeur pour lui vendre certains ordinateurs domestiques que vous avez réussi à dénicher. Ce vendeur vous annonce que le sergent détective est à la recherche d'ordinateurs volés. Cette nouvelle, évidemment, fait immédiatement baisser les prix.

Les problèmes d'Arthur se compliquent encore plus si le sergent détective Chisholm surgit à l'entrepôt — où Arthur stocke ses marchandises — et trouve la marchandise volée. Afin de conclure des ventes et de trouver Terry, dont vous avez besoin pour transporter les marchandises, vous devez souvent aller au bar Winchester. Dans cet endroit, comme dans de nombreux autres dans le jeu, vous pouvez avoir jusqu'à six interlocuteurs potentiels. Leurs visages apparaissent dans l'un des cadres de l'écran, et, afin d'approcher l'un d'eux, vous n'avez qu'à appuyer sur la touche qui correspond à ce personnage.

L'approche de personnages au Winchester n'est nécessaire que si vous voulez parler à quelqu'un en particulier, comme Dave le barman ou Terry — la plupart des autres essaient de vous vendre des marchandises diverses. La stratégie ici est un peu la même que lors d'une tentative de vente. Vous constatez d'abord le prix demandé pour les marchandises, la quantité disponible, puis vous commencez à marchander. Tout en négociant le prix, l'horloge égrène les minutes devant vous — le contact démarre si l'affaire n'est pas conclue après un certain temps.

Terry est un personnage plutôt mobile et il est parfois difficile de le joindre; lorsqu'il exécute un travail, il s'attend à être récompensé — qu'on lui paie un verre, qu'on lui verse quelque somme d'argent. Par conséquent, lors de la négociation d'une affaire, vous devez garder à l'esprit que les bénéfices doivent être suffisants pour payer Terry.

Sans aucun doute, la partie la plus amusante de ce jeu est la négociation. Pendant que vous discutez du prix avec votre adversaire, l'écran affiche des commentaires de ce genre : « bonne affaire », « qualité fantastique ». Lors de l'offre d'un prix, il est important de taper la bonne syntaxe, comme « j'offre 20 balles ». Sinon l'ordinateur ne peut vous comprendre.

Contrairement à de nombreux jeux de cette nature, Minder est aussi amusant et intéressant que la série télévisée. Il a très bien réussi à recréer l'atmosphère de l'émission et devrait remporter autant de succès chez les amateurs de jeux que chez les personnes qui ont apprécié la série télévisée.

Minder : pour Sinclair Spectrum, MSX, Memotech, Amstrad et Commodore 64.

Éditeurs : DK'tronics.

Auteur : Don Priesley.

Format : cassette.

Manche à balai : non nécessaire.

Page manquante
(publicité)

Page manquante
(publicité)